COMPUTER LANGUAGE™

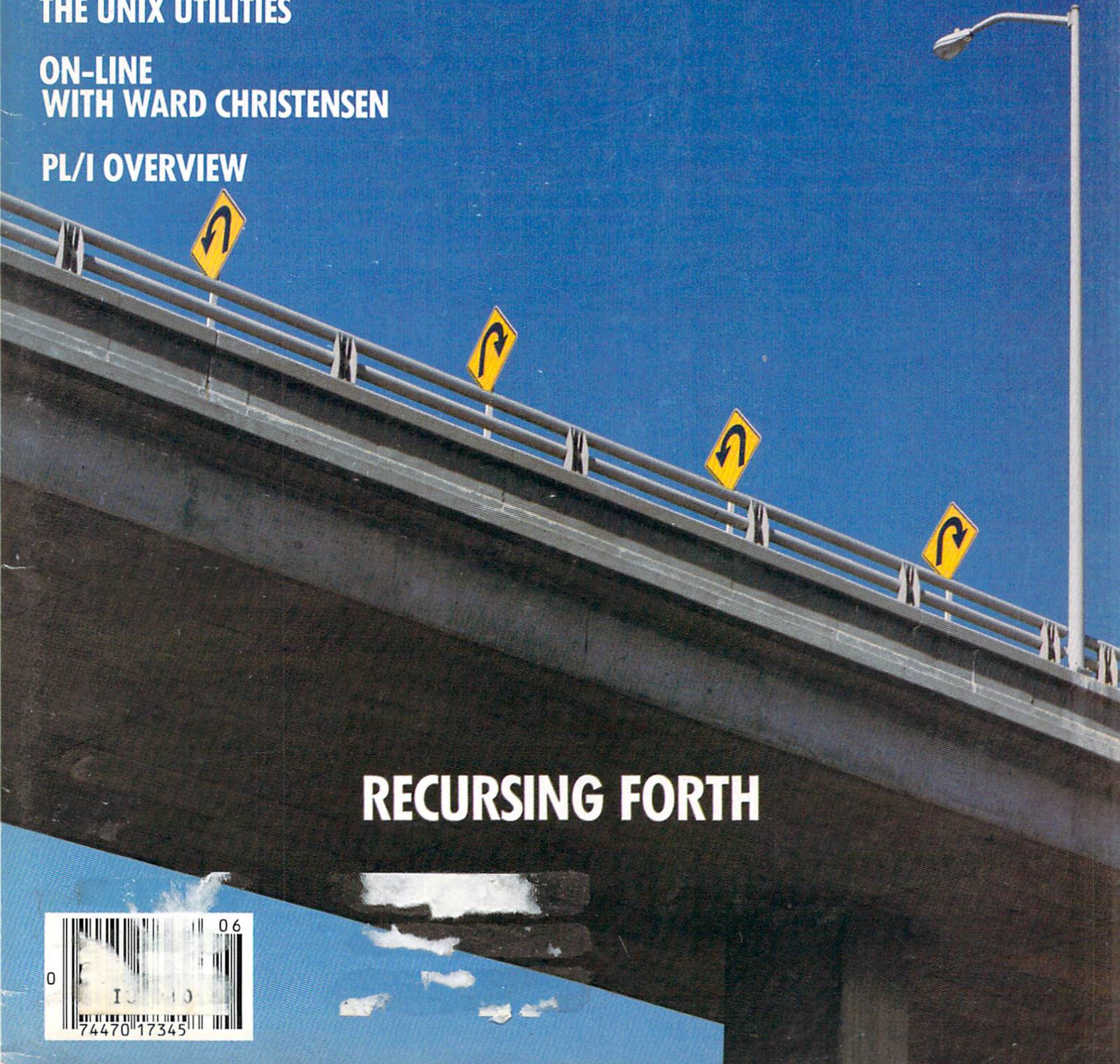
$2.95
Canada $3.95

VOLUME 2, NUMBER 6
JUNE 1985

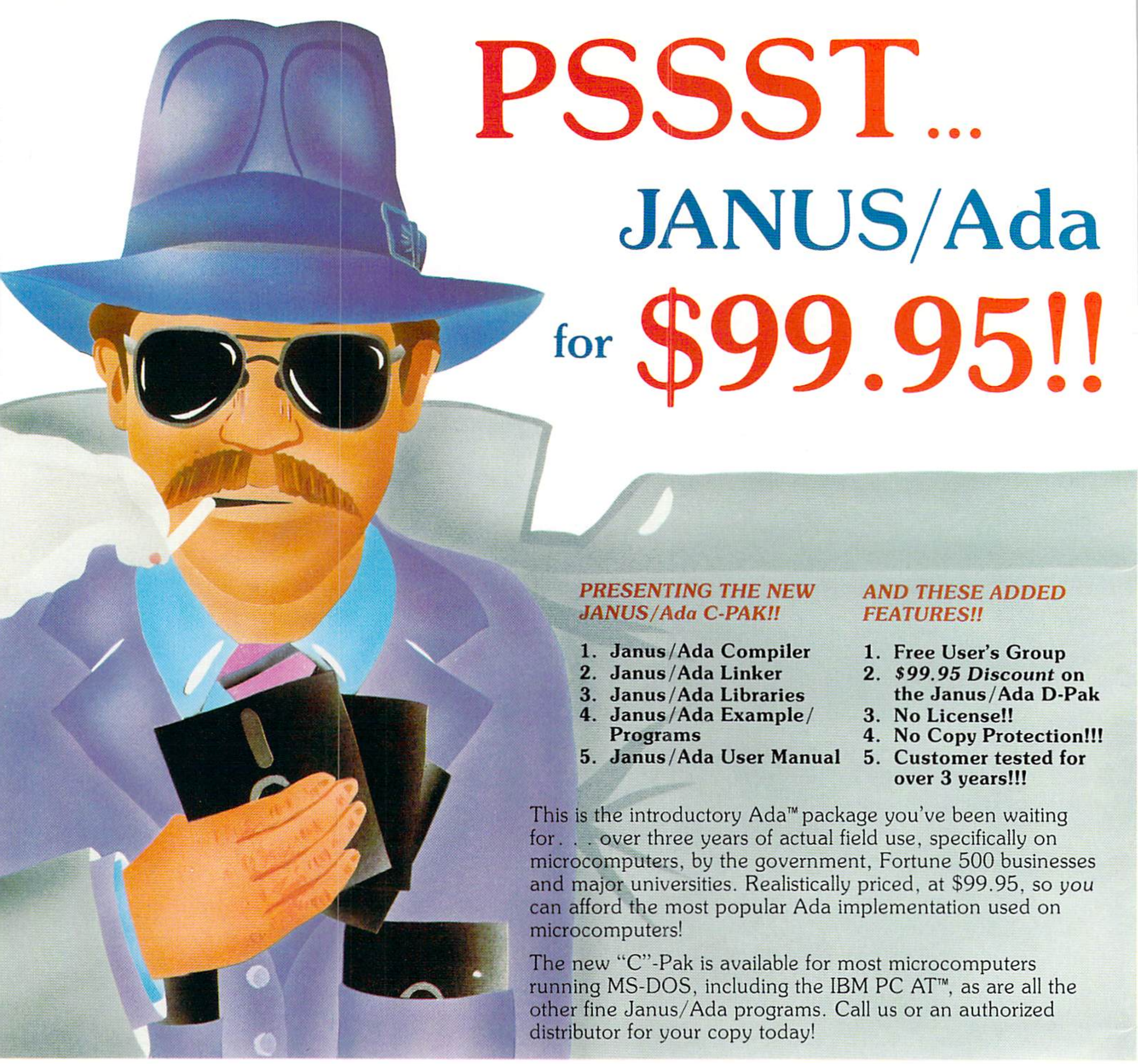
PORTING THE UNIX UTILITIES

ON-LINE WITH WARD CHRISTENSEN

PL/I OVERVIEW



RECURSING FORTH



PSSST...

JANUS/Ada for $99.95!!

PRESENTING THE NEW JANUS/Ada C-PAK!!

1. **Janus/Ada Compiler**
2. **Janus/Ada Linker**
3. **Janus/Ada Libraries**
4. **Janus/Ada Example/ Programs**
5. **Janus/Ada User Manual**

AND THESE ADDED FEATURES!!

1. **Free User's Group**
2. ***$99.95 Discount* on the Janus/Ada D-Pak**
3. **No License!!**
4. **No Copy Protection!!!**
5. **Customer tested for over 3 years!!!**

This is the introductory Ada™ package you've been waiting for . . . over three years of actual field use, specifically on microcomputers, by the government, Fortune 500 businesses and major universities. Realistically priced, at $99.95, so *you* can afford the most popular Ada implementation used on microcomputers!

The new "C"-Pak is available for most microcomputers running MS-DOS, including the IBM PC AT™, as are all the other fine Janus/Ada programs. Call us or an authorized distributor for your copy today!

National Distributors

Westico, Inc.
25 Van Zant St.
Norwalk, CT 06855
(203) 853-6880

ASH II
7407 Marisol
Houston, TX 77083
(713) 933-1828

A.O.K. Computers
816 Easley St., Suite 615
Silver Springs, MD 20910
(301) 588-8446

Trinity Solutions
5340 Thornwood Dr., Suite 102
San Jose, CA 95123
(408) 226-0170

MicroProgramming, Inc.
P.O. Box 3356
Chatsworth, CA 91313
(818) 993-6475

International Distributors

Ada Australia
218 Lutwyche Rd.
Windsor 4030
QLD. Australia
(07) 57 9997

Progesco
155, rue du Faubourg
St. -Denis
75010 Paris
France
(1) 205.39. 47

Lifeboat, Inc. Japan
3-6, Kando-Nishikicho
Chiyoda-ku
Tokyo 101, JAPAN
03-293-4711

CP/M, CP/M-86, CCP/M-86 are trademarks of Digital Research, Inc.
*ADA is a trademark of the U.S. Department of Defense
MS-DOS is a trademark of Microsoft
©Copyright 1984 RR Software

RR SOFTWARE, INC.

specialists in state of the art programming

P.O. Box 1512 Madison, Wisconsin 53701
(608) 244-6436 TELEX 4998168

CIRCLE 58 ON READER SERVICE CARD

Software development isn't a mountainous task once you eliminate the high C errors.

When you can find and fix bugs at the earliest possible moment, creating software stops being such an uphill grind.

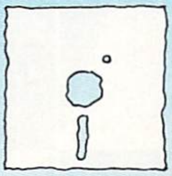
And the Smart/C Environment makes it possible. It's a complete, fully-integrated development environment for C that saves you from the creativity-inhibiting cycle of edit, compile, re-edit, re-compile, link, load, test, re-edit, re-compile, etc., ad infinitum. Smart/C puts the fun back in programming, because you spend your time creating... not waiting.

Here's why. Syntax errors are eliminated automatically as code is entered. Smart/C's highly integrated editor and interpreter allow you to interpret your program at any time in the creation process, so logic errors can be ferreted out as soon as the algorithm exists—long before any compile, link, or load.

The complete integration of the editor and interpreter means you can stop anywhere in the interpret cycle, edit, and then go right back into the interpreter exactly where you left off. Not only that, the screen-oriented user interface lets you see all operations, even interpretation, right on the listing of the code.

And to make maintenance programming easier, Smart/C's Migrator allows existing C code produced with any editor to be modified and run within the Smart/C Environment.

All of which makes Smart/C an excellent tool. It's flexible, non-restrictive, and lets you create elegant, readable, error-free programs that you can watch run with a great feeling of satisfaction.



Smart/C™

Free Demo Disk!

To fully appreciate Smart/C, you have to see it in action. For your free IBM PC MS-DOS demo disk, call us. Or write us on your company letterhead.

AGS Computers, Inc., Advanced Products Division,
1139 Spruce Drive, Mountainside, NJ 07092.
800-AGS-1313. In NJ, 201-654-4321.

Smart/C Features

The Smart/C Environment
- ☐ Fully integrated editor and interpreter
- ☐ Only one load brings them both in
- ☐ One command set
- ☐ Move between one another at will

Syntax Directed Editor
- ☐ vi-like command set
- ☐ Automatically provides formats for blocks, *for, case* and *if* statements

Interpreter
- ☐ Current module can call external modules during interpretation
- ☐ Has Include capability
- ☐ Totally precompilation—no incremental compile
- ☐ Can interpret partially defined files allowing for rapid prototyping
- ☐ Variable speed of interpretation
- ☐ Multiple windows with user-defined sizes

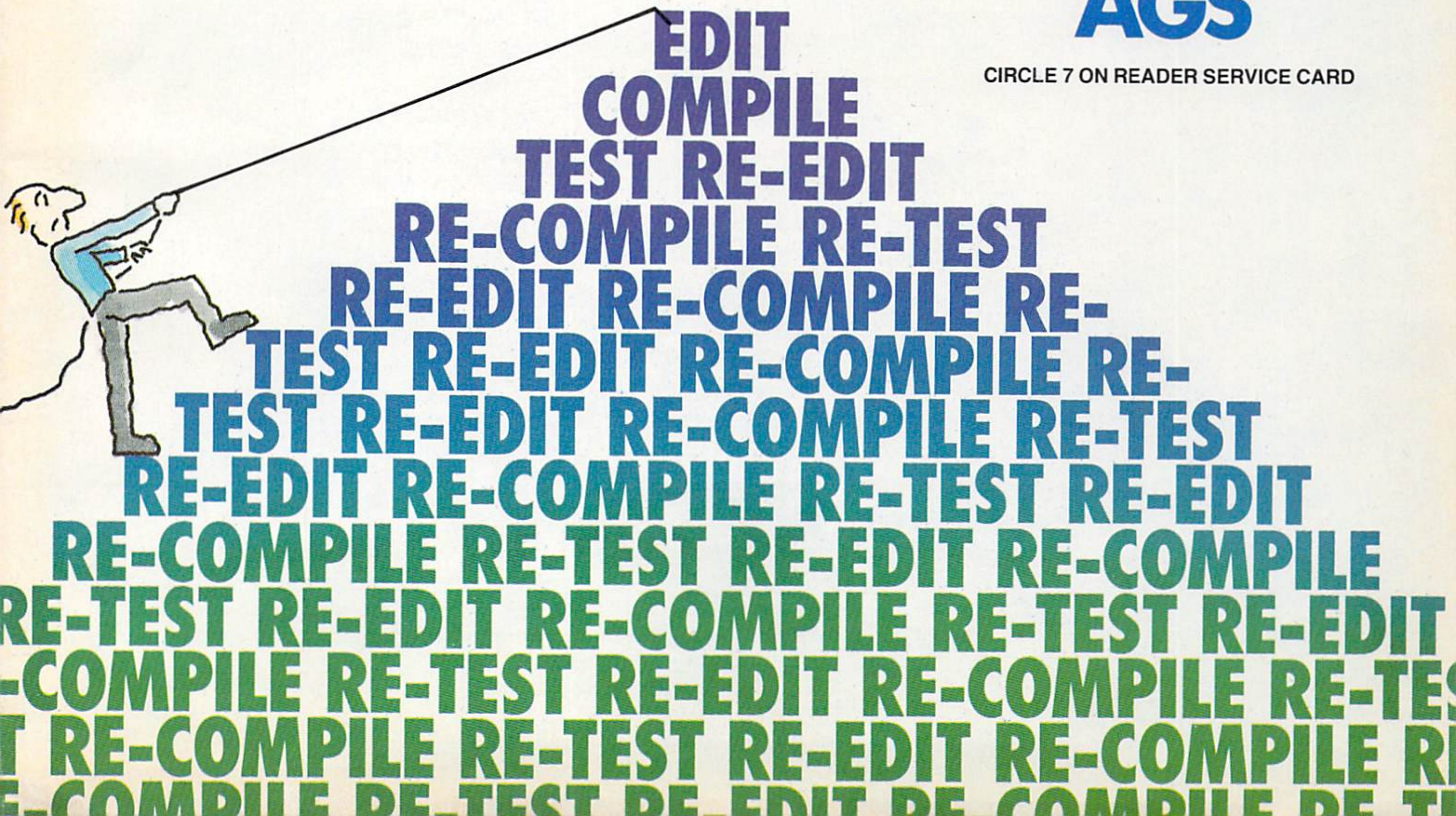
The Smart/C Migrator
- ☐ Allows C code produced with any editor to be interpreted by Smart/C
- ☐ Reformats for readability

Smart/C has been ported to UNIX™ System V Release 2, Berkeley 4.2, Xenix,™ and MS-DOS. Versions run on 8086- and 68000-based machines, as well as proprietary architectures. Smart/C runs on PCs, micros, supermicros, minis, and even mainframes.

Trademarks—Smart/C: AGS Computers, Inc.; UNIX: AT&T Bell Labs; Xenix and MS-DOS: Microsoft Corp; IBM PC: IBM Corp.

AGS

CIRCLE 7 ON READER SERVICE CARD


EDIT
COMPILE
TEST RE-EDIT
RE-COMPILE RE-TEST
RE-EDIT RE-COMPILE RE-
TEST RE-EDIT RE-COMPILE RE-
TEST RE-EDIT RE-COMPILE RE-TEST
RE-EDIT RE-COMPILE RE-TEST RE-EDIT
RE-COMPILE RE-TEST RE-EDIT RE-COMPILE
RE-TEST RE-EDIT RE-COMPILE RE-TEST RE-EDIT
-COMPILE RE-TEST RE-EDIT RE-COMPILE RE-TE
T RE-COMPILE RE-TEST RE-EDIT RE-COMPILE R

Clipper gives dBASE III™ users more time to do more. Or less.

Clipper™ allows you to run all dBASE III™ programs 2 to 20 times faster than they do with the standard dBASE interpreter.

That frees up extra time you're wasting if you're running dBASE III programs without Clipper.

Extra time to think. To create. To produce. To use as you choose.

You see, Clipper is the first true compiler for dBASE III. Clipper eliminates the time-consuming translation which the dBASE interpreter performs line after line whenever a program is run.

With Clipper, once you've debugged your source code, it's compiled into more efficient machine code.

And Clipper compiles all your dBASE III programs. The ones you have today. The ones you'll have tomorrow. But don't wait until tomorrow to order Clipper.

Today, Clipper has already been purchased to speed up dBASE run time at 3M and Touche Ross. At Exxon and NASA. In the Harvard Physics Department. For the State of Arizona and TRW.

And that's just a few of the installations worldwide. From Greece to Venezuela to Canada to Europe.

So stop wasting time. Call our toll-free 800 number and get Clipper.

You'll spend less time running dBASE III and more time running the rest of your life.

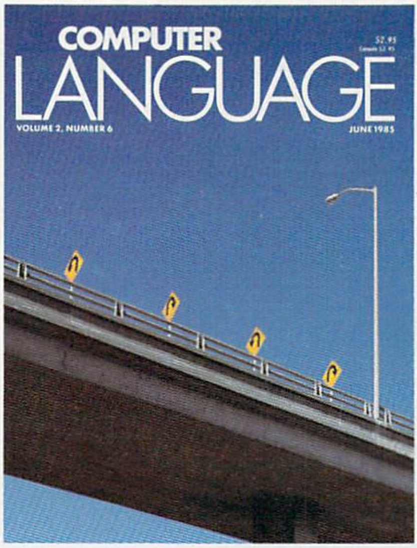
Nantucket™

5995 S. Sepulveda Blvd., Culver City, CA 90230 (800) 556-1234 ext. 225 In California (800) 441-2345 ext. 225

CIRCLE 59 ON READER SERVICE CARD

dBASE III is a registered trademark of Ashton Tate

COMPUTER LANGUAGE



ARTICLES

Forth: Twithe Curthed, too! — **27**
by Jean-Pierre Schachter
Recursion is often an attractive alternative to iteration, especially in cases where the original task can be broken down into a finite set of smaller, similar tasks. While some implementations of Forth have recursive control structures built in, many do not. This article presents a method for adding recursion to Forth by taking advantage of Forth's inherent extensibility, thereby allowing Forth programmers the same recursive powers that were built into C and LISP.

Porting the UNIX Utilities — **33**
by Alan Filipski
UNIX system utilities are commonly dismissed as being source-code portable since they were written in the C language. As a general statement, this is true, but there are exceptions. Filipski describes the hurdles he and a team of coworkers had to leap when converting UNIX System V utilities from a VAX to an M68000-based computer.

Symphony Command Language — **41**
by Darryl Rubin
Many top-selling programs owe their success to their user programmability. With the new breed of integrated programs like Lotus's Symphony, an embedded language can do more than create macros—it can offer a complete development environment. This article presents the Symphony Command Language and describes some interesting macros you can use right away.

Getting to Know PL/I — **51**
by Gary Sarff
Compared to the lean elegance of Pascal, PL/I is considered by some purists of language design to be a monstrosity. But PL/I is a quite useful and powerful language for business and scientific languages. In this article we explore the philosophy and feel of PL/I and study its strengths and limitations.

DEPARTMENTS

Publisher's Notes — 5
Feedback — 9
CrossThoughts — 13
Binary trees and B-trees
Designers Debate — 21
The dilemma of copy protection
Public Domain Software Review — 59
More Macintosh software
Exotic Language of the Month Club — 65
COMAL: A friendly micro language
ComputerVisions — 71
On-line with Ward Christensen
Product BINGO — 81
Software Reviews — 83
Computer Helper's ConIX, System Guild's CSHARP and CGRAPH, Phoenix's Pfix86, Plink86, and Plink86 Plus
Advertiser Index — 96

Professional Pascal™

Smaller & Faster code Plus 22 reasons for not using Microsoft® Pascal

The List:

Compile Time
Symbol table space limited only by available memory
Cross referencer
ANSI–standard checking
Option to search a list of directions to find any input file
Source interleaved with Object in Object listing
Full Intel OMF Line/Symbol/Type records for debugger
Run Time
Code quality
Five models of memory on the 8086: small, medium, compact, big and large
Code produced suitable for ROM
Expressions
Long/short integer mixes
Set constant expressions
Set elements in the range $-2^{31}..2^{31}-1$ or 0..256
Type casts
// = Concat for concatenation of all strings
Statements
Iterators (for data abstraction from CLU)
Assignment to string variables of differing lengths
Declarations
Otherwise (or else) in record declarations
Intrinsics
Readln(X:M)
Inc, Dec, Incl, Decl, from Modula–2
Data Types
Comparison (for equality) of record or array types
The value -32768 allowed as an integer
Maximum size of a data structure on the 8086

Information on Microsoft Pascal obtained from the 1984 release 3.2. printing of the MS manual.

Microsoft has a problem. The 22 features listed above are missing from MS Pascal.

Microtec Research has the answer. Professional Pascal.

Professional Pascal is more than just powerful features. It generates the most efficient code you've ever seen for 8086/8087/8088/186/188/286/287 in 5 memory models.

Actually 35 positive differences exist in our favor.

In a direct comparison – there isn't any!

We're ***Functional*** *and* ***Fast*** *and* ***Serious*** *about our products. We've been providing flexible and economical solutions for software developers since 1974.*

Beginning with product concept, through development, quality assurance, and post-sales support ***– Quality, Compatibility*** *and* ***Service*** *are the differences which set Microtec Research apart from others.*

If you're a serious software developer, shopping for software development tools, call or write today for more information. ***800-551-5554*** *In CA call* ***(408) 733-2919***

No matter what Pascal or chip you choose, recompile with the fully compatible, across all implementations, Professional Pascal.

We were never accused of humility – only solid software development tools.

MICROTEC®
RESEARCH

CIRCLE 36 ON READER SERVICE CARD

3930 Freedom Circle, Suite 101, Santa Clara, CA 95054 Mailing Address: P.O. Box 60337, Sunnyvale, CA 94088

Professional Pascal is a trademark of MetaWare, Inc.

Microtec is a registered trademark of Microtec Research, Inc.

Microsoft is a registered trademark of Microsoft Corporation.

Publisher's Notes

This month's *COMPUTER LANGUAGE* is a bit of a novelty in computer publishing today—it is *not* a special issue. It has no special theme or specific product wrap-up. We feel it is a refreshing change to be able to offer you a wide variety of subjects.

The reader response we've received about our theme product wrap-ups covering C, COBOL, and BASIC has been very positive. This format allows us to cover the maximum amount of products in a fair comparison. Unfortunately, we cannot go into depth on each product as we do this month with four individual reviews. Which approach to product reviews is most useful to you?

COMPUTER LANGUAGE has celebrated an anniversary of sorts. We exhibited at the West Coast Computer Faire for the second time. This year was quite different than last for us. Last year we snuck into the show and shared a booth with another exhibitor. We had no money and just an idea for a magazine. Fortunately some terribly trusting people bought subscriptions.

This year we actually had magazines and they sold like hot cakes. The Faire was an incredible amount of fun and gave us a chance to meet many loyal readers face to face.

We mention this show because it really is the only major trade show that still caters to the programmer. Many of the old-time exhibitors complained about the new show management and location. But you've got to expect changes after eight years. No one ever said progress is for the better. Hey, any computer show where the number of people walking around in jeans outnumbers the people in three-piece blue suits can't be all bad.

The initial reaction to the *COMPUTER LANGUAGE* C Seminar in Cambridge, Mass., has been fantastic. We've received a great many reservations after officially announcing it just a month ago. Reserve your space today and save $100 off the registration price by signing up by June 30th. See page 93 for details.

And now for this month's potpourri of articles. Jean-Pierre Schachter's "Forth: Twithe Curthed, too," shows that Forth can be given the same recursive power as LISP and C. "Porting the UNIX Utilities," Alan Filipski's third article for *COMPUTER LANGUAGE*, discusses the problems the author encountered when porting UNIX utilities from a VAX to an M68000-based system.

"Symphony Command Language" by Darryl Rubin, another popular *COMPUTER LANGUAGE* author, covers Symphony, Lotus's combination spreadsheet, data base, word processor, graphics, and communications package. Gary Sarff's "Getting to Know PL/I" provides insight into this powerful but not always well-known language.

Our usual departments also cover a wide range of topics—from B-trees to copy protection, the exotic language COMAL to an interview with MODEM7 creator Ward Christensen, and more.

Carl Landau

Carl Landau
Publisher

Telecommunicate to COMPUTER LANGUAGE

COMPUTER LANGUAGE has established two bulletin board systems for you to upload and download text and binary programs, as well as to leave your own electronic Letter to the Editor. All the program listings referred to in every issue of the magazine will be available here.

For those readers without access to a modem who desire a copy of program listings referred to but not printed in an issue, send $5 to *COMPUTER LANGUAGE*, attention: Listings Dept., 131 Townsend St., San Francisco, Calif. 94107. We will mail you a copy of all the listings not printed in this issue.

In addition, *COMPUTER LANGUAGE* has its own Special Interest Group on CompuServe's national data base. After calling into your local CompuServe node, simply type "GO CLM" at any prompt and you'll be in our SIG.

To access our bulletin board, set your computer or terminal to the following parameters: 8 data bits, no parity, 1 stop bit, full duplex, and either 300 or 1200 baud. The telephone number is (415) 957-9370. After your modem makes the connection, type RETURN several times, and everything else is easy.

Both systems are open 24 hours per day, 7 days per week. Due to the heavy number of callers, please do not log into the system more than one time per day. Messages left on either system will be combined the following day.

COMPUTER
LANGUAGE

EDITOR
Craig LaGrow

MANAGING EDITOR
Regina Starr Ridley

TECHNICAL EDITOR
John Halamka

PRODUCT REVIEW EDITOR
Hugh Byrne

EDITORIAL ASSISTANTS
Kathy Kincade, John Harrington

CONTRIBUTING EDITORS
Mark Brown, Doug Millison, Tim Parker,
Namir Clement Shammas, Ken Takara

INDUSTRY NEWS CONSULTANT
Bruce Lynch

SPECIAL PROJECTS MANAGER
Jan Dente

OPERATIONS CONSULTANT
Beatrice C. Blatteis

CIRCULATION COORDINATOR
Renato Sunico

ART DIRECTOR
Jeanne Schacht

COVER PHOTO
Dow/Clement Photography

PRODUCTION ARTISTS
Anne Doering, Barbara Luck

PRODUCTION
Steve Campbell, Kyle Houbolt

TECHNICAL CONSULTANT
Addison Sims

ACCOUNTING MANAGER
Lauren Kalkstein

WHOLESALE COORDINATOR
Nicola Sullivan

PUBLISHER
Carl Landau

COMPUTER LANGUAGE *(ISSN 0749-2839) is published monthly by* COMPUTER LANGUAGE *Publishing Ltd., 131 Townsend St., San Francisco, CA 94107. (415) 957-9353.*

Advertising: For information on ad rates, deadlines, and placement, contact Carl Landau at (415) 957-9353, or write to: COMPUTER LANGUAGE*, 131 Townsend St., San Francisco, CA 94107.*

Editorial: Please address all letters and inquiries to: Craig LaGrow, Editor, COMPUTER LANGUAGE*, 131 Townsend St., San Francisco, CA 94107.*

Subscriptions: Contact COMPUTER LANGUAGE*, Subscriptions Dept., 2443 Fillmore St., Suite 346, San Francisco, CA 94115. Single copy price: $2.95. Subscription prices: $24.95 per year (U.S.); $30.95 per year (Canada and Mexico). Subscription prices for outside the U.S., Canada, and Mexico: $36.95 (surface mail), $54.95 (air mail)—U.S. currency only. Please allow six weeks for new subscription service to begin.*

Postal information: Second-class postage is paid at San Francisco, CA and at additional mailing offices.

Reprints: Copyright 1985 by COMPUTER LANGUAGE *Publishing Ltd. All rights reserved. Reproduction of material appearing in* COMPUTER LANGUAGE *is forbidden without written permission.*

Change of address: Please allow six weeks for change of address to take effect. POSTMASTER: Send change of address (Form 3579) to COMPUTER LANGUAGE*, 131 Townsend St., San Francisco, CA 94107.*

COMPUTER LANGUAGE *is a registered trademark owned by the magazine's parent company, CL Publications. All material published in* COMPUTER LANGUAGE *is copyrighted © 1985 by CL Publications, Inc. All rights reserved.*

SuperKey
TOP SECRET
ROBERT
85 TINNEY

Borland Does It Again: SuperKey $69.95

Sure, ProKey™ is a nice little program. But when the people who brought you Turbo Pascal and SideKick get serious about keyboard enhancers, you can expect the impossible . . . and we deliver.

	ProKey	SuperKey
ALL FEATURES RESIDENT IN RAM AT ALL TIMES	NO	YES
RESIDENT PULL-DOWN MACRO EDITOR	NO	YES
RESIDENT FILE ENCRYPTION	NO	YES
PROKEY COMPATIBILITY	YES	YES
DISPLAY PROTECTION	NO	YES
ABILITY TO IMPORT DATA FROM SCREEN	NO	YES
PULL-DOWN MENU USER INTERFACE	NO	YES
CONTEXT-SENSITIVE ON-LINE HELP SYSTEM	NO	YES
DISPLAY-ONLY MACRO CREATION	NO	YES
ENTRY AND FORMAT CONTROL IN DATA FIELDS	NO	YES
COMMAND KEYS REDEFINABLE "ON THE FLY"	NO	YES
PRICE	129^{95}	69^{95}

Total ProKey compatibility. Every Prokey Macro file may be used by SuperKey *without change* so that you may capitalize on all the precious time you've invested.

Now your PC can keep a secret! SuperKey includes a resident file encryption system that uses your password to encrypt and decrypt files, even while running other programs. Two different encryption modes are offered:

1. Direct overwrite encryption (which leaves the file size unchanged) for complete protection. At no point is a second file that could be reconstructed by an intruder generated. Without your secret password, no one will ever be able to type out your confidential letters again!

2. COM or EXE file encryption which allows you to encrypt a binary file into an ASCII file, transmit it through a phone line as a text file and turn it back again into an executable file on the target machine (only of course if your correspondent knows the secret password!). Now, you will even be able to secretly exchange programs through Public Bulletin Board Systems or services such as CompuServe.

Totally memory resident at all times, gives SuperKey the ability to create, edit, save and even recall new or existing macro files anytime, even while running another program.

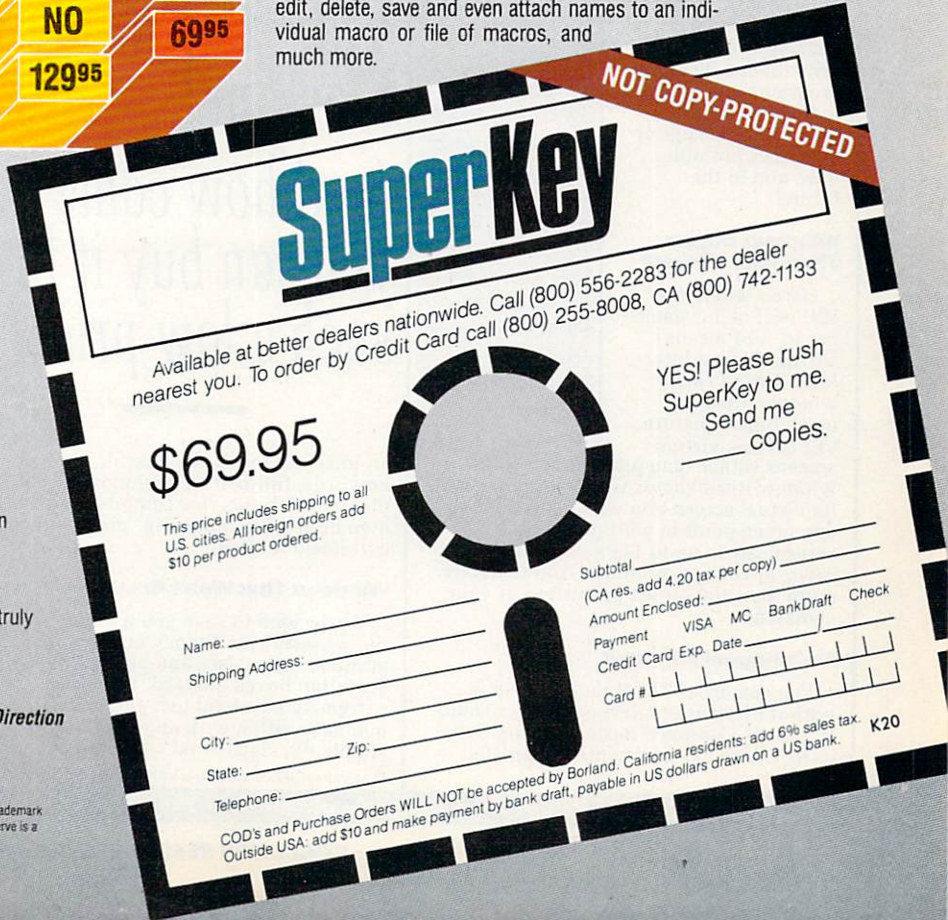
Pull down macro editor. Finally, a sensible way to create, edit, change and alter existing macro definitions. Even while using another application, a simple keystroke instantly opens a wordprocessor-like window where you're allowed to see, edit, delete, save and even attach names to an individual macro or file of macros, and much more.

Sorry ProKey !

Superb software at reasonable prices!
There is much more to SuperKey. Maybe the best reason to buy SuperKey is that it is a Borland International Product. Each one of our products is the best in its category. We only believe in absolutely superb software at reasonable prices!

An offer you can't refuse.
Whether you are a ProKey user or you've never used a keyboard enhancer before, your boat has come in. You can get your copy of SuperKey at this irresistible price.

Get your PC a SuperKey today!
SuperKey is available now for your IBM PC, XT, AT, jr. and truly compatible microcomputers.



NOT COPY-PROTECTED

SuperKey

Available at better dealers nationwide. Call (800) 556-2283 for the dealer nearest you. To order by Credit Card call (800) 255-8008, CA (800) 742-1133

$69.95

This price includes shipping to all U.S. cities. All foreign orders add $10 per product ordered.

YES! Please rush SuperKey to me. Send me ____ copies.

Name: ____
Shipping Address: ____
City: ____
State: ____ Zip: ____
Telephone: ____

Subtotal ____
(CA res. add 4.20 tax per copy) ____
Amount Enclosed: ____
Payment VISA MC BankDraft Check
Credit Card Exp. Date ____
Card # ____

K20

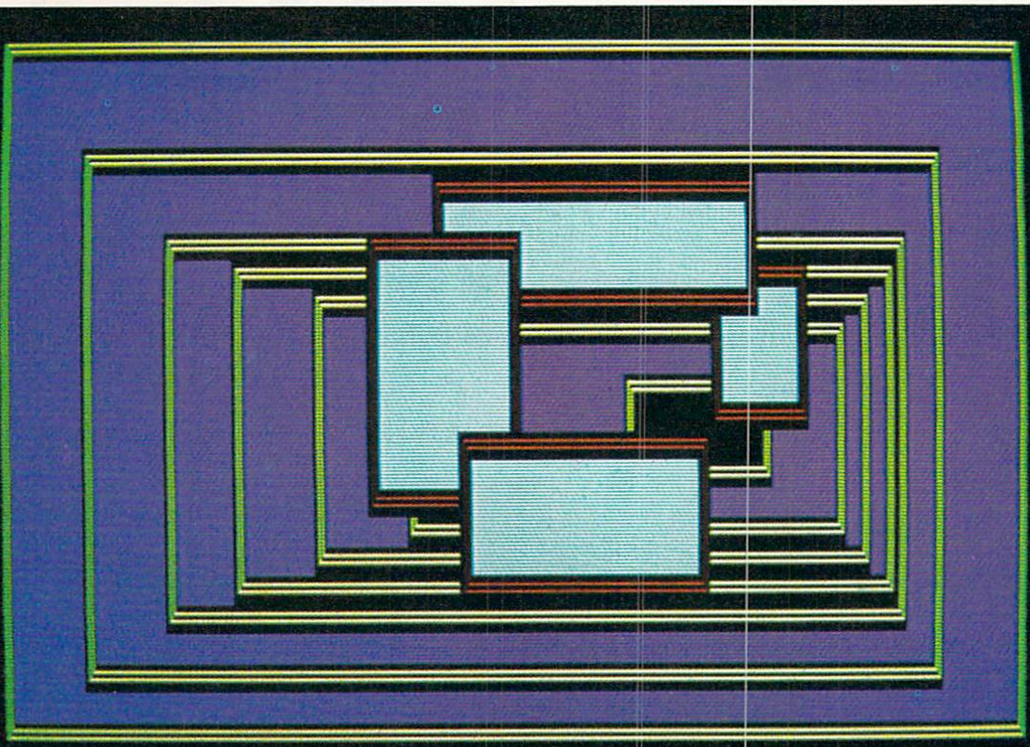
COD's and Purchase Orders WILL NOT be accepted by Borland. California residents: add 6% sales tax. Outside USA: add $10 and make payment by bank draft, payable in US dollars drawn on a US bank.

BORLAND INTERNATIONAL

Software's Newest Direction
4585 Scotts Valley Drive
Scotts Valley, CA 95066
TELEX 172373

IBM is a registered trademark of International Business Machine Corporation. ProKey is a trademark of RoseSoft. SuperKey and SideKick are trademarks of Borland International, Inc. CompuServe is a trademark of CompuServe Corp.

CIRCLE 9 ON READER SERVICE CARD



WHEN YOU BUILD A HOUSE . . . YOU DON'T NEED TO MAKE THE WINDOWS YOURSELF. NOW . . . THE SAME IS TRUE WHEN YOU'RE WRITING CODE.

Why did Simon & Schuster, 3Com, Tymshare, and Revlon choose VSI – The Window Machine?

(and how come you can buy it for such a low price?)

$59.95

Windows With A View Toward The Future

The Window Machine™ occupies only 12K! Written in tight, fast Assembler, it performs like a racing engine... with more power than you'll probably ever need. Yet, it's an engine designed to fit in the vehicle of your choice... from a "stripped-down" 128K IBM PC to a fully loaded AT. The programs you write today will run on the broadest range of machines possible... now, and in the future.

Windows Bigger Than Your Screen?

Here's where the VSI part of our name fits in. VSI means Virtual Screen Interface. Behind each window, there's a much bigger picture. VSI defines virtual screens rather than just windows. The window itself shows whatever portion of its virtual screen you wish to exhibit at any given point in your program. Each screen can be up to 128 x 255 (columns x rows, or rows x columns). And there are more than 100 screen primitives at your command.

These are coders' windows... designed to be built into the programs you are writing. They can overlap, move anywhere on the screen, grow, shrink, vanish or blink. They can be bordered in anything from a simple line to flashing asterisks... or even no border at all. And you can have up to 255 of them at a time! Color or monochrome ...of course!

Multilingual Windows

You can order The Window Machine with the language interface of your choice: C, Pascal, Compiled Basic, Fortran, Cobol, or PL1. We've even recently completed an interface for Turbo Pascal*, so that now true, full-featured windowing can be utilized with this fine compiler. (Turbo's own built-in "windowing" procedure is extremely limited).

Windows That Won't Break You

We decided to save you a lot of money. So, we left behind fancy binders, monogrammed slip cases and plastic presentation boxes. Instead, you'll find an extremely powerful tool and a 200 page manual written with an eye toward simplicity, clarity and completeness. (We figured if you wanted ribbons and bows you could always add them yourself.)

And by offering you the product ourselves, we were able to cut out all the middlemen and save you a tremendous amount of money.

VSI *THE WINDOW MACHINE*

Available for the IBM PC, XT, AT, IBM Compatibles, Wang, T.I., and HP 150

The Window Machine Includes:

- *Zoom Windows*
- *Multiple Virtual Screens (up to 255)*
- *Choice of Borders (including flashing borders)*
- *Support for all Color and Monochrome Video Attributes (no graphics card required)*
- *Built-in Diagnostics*
- *And much, much more*

ORDER YOUR COPY OF
VSI—THE WINDOW MACHINE TODAY
For Visa & MasterCard orders call toll free:
800-536-8157 Ext. 824 In CA 800-672-3470 Ext. 824
Call Mon.-Fri. 6A.M. to 12P.M., Sat. & Sun. 6A.M. to 8P.M. (P.S.T.)

The Window Machine™ $59.95 + $5 Shipping and Handling
LANGUAGE INTERFACE:
☐ Lattice C ☐ Realia Cobol ☐ Microsoft Basic Compiler ☐ Microsoft Fortran
☐ PL1 ☐ Microsoft Pascal ☐ Turbo Pascal (full featured true windowing)

COMPUTER ____________

Name ____________

Address ____________

City ____________ State ____ Zip Code ____________

☐ Check ☐ Money Order ☐ VISA ☐ MasterCard

Card # ____________ Exp. Date ____________

*California residents: tax included. Orders outside USA: Please add $10 for shipping and handling

30 day Money Back Guarantee

CL

AMBER
AMBER SYSTEMS
1171 S. Saratoga-Sunnyvale Road
San Jose, CA 95129

*Turbo Pascal is a Trademark of Borland International

AMBER SYSTEMS, INC. 1171 S. Saratoga-Sunnyvale Road, San Jose CA 95129

FOR DEALER INQUIRIES: CALL OUR 800 NUMBER

CIRCLE 2 ON READER SERVICE CARD

FEEDBACK

Illustration: Anne Doering

Just what doc ordered?

Dear Editor:

I was most interested to read Peter Reece's article on APT (*COMPUTER LANGUAGE*, Apr. 1985, pp. 43-55). As a novice (barely) Forth and LISP programmer, I am intrigued by the thought of a language that combines constructs of both of these. I was also quite interested to see the medical examples. Has the author been involved in artificial intelligence applications in the medical field? As a physician and computer enthusiast, I have been most interested in this area, and I would greatly appreciate the chance to look at APT when the language is available.

Zachary T. Bloomgarden, M.D.
New York, N.Y.

Anxious for APT

Dear Editor:

I am inquiring as to the progress and availability of the APT language. Though I am no computer genius, I do have a knack for getting the obstinate machine to do what I want. Having programmed applications primarily in BASIC, with some work in APL, I have been looking for a more suitable tool for my applications—instructional software making extensive use of graphics.

Any information regarding APT, its availability, documentation, and costs would be greatly appreciated. If APT is as good as it appears, I may want to use it for all of my programming needs. I can't wait.

Ed Dickerson
Garrison, Iowa

Author Peter Reece responds: Yes, I have been involved in artificial intelligence in the medical field—specifically, working on the creation of artificial ears from an AI point of view.

I hope to have APT available (at minimum cost) on disks for the IBM PC in about six months. Because I am expanding the language—including adding quite sophisticated windowing and expanded AI capabilities—I feel some reluctance about releasing the Radio Shack version at this time. I also want to make sure there is a proper manual and adequate documentation.

If you are interested in corresponding further on APT or in acquiring the language when it is available, please write to me, Peter Reece, c/o COMPUTER LANGUAGE, *131 Townsend St., San Francisco, Calif. 94107.*

Also, because I have found out that another company is using the name APT, the language has been renamed RAPT.

A simple sort

Dear Editor:

"Sorting by Dispersion" by David Keil (*COMPUTER LANGUAGE,* Apr. 1985, pp. 27-32) recalled a simple (minded) sort algorithm that occurred to me lately. It must have been noted somewhere before, but there will be no rush to take credit. Since it can be described shortly over the telephone, perhaps it should be called Telephone Sort—or perhaps just Simple Sort.

Given an array of elements $A(1{:}m)$, indexes i and j, and a counter c, proceed as is shown in Listing 1. This takes m^2 comparisons and m interchanges. Attempts to improve it mutate it into more familiar algorithms.

The only advantage I know is that the inner loop can be realized with a single instruction on some array processors and, of course, could be easily microcoded. And it might be useful in programming over the telephone without a modem.

M.B. Clausing
Syosset, N.Y

A transnational KISS

Dear Editor:

As a recent subscriber, I was taken aback by Namir Shammas's new language project. Who on earth needs yet another computer language? But on second thought, every computernik worth his salt must have at least one compiler and one language to his credit. So why not take all these great ideas and pick the best nuggets from them all—something worthwhile might result after all.

My own contribution is a minimal language called KISS (no, the second S stands for Sir or Sister, as the case may be). The language uses no key words and is therefore truly transnational. Here is the instructional manual:

( )	begin end
|X|	absolute value of X
-> X	get X
<- X	put X
@	any permissible value
? E	proceed if E is true
! N	repeat N times

The loop statement *!N* is a default form for *! c1, c2 . . . N*, where $c1$ is the first counter value, $c2-c1$ the increment, and N the inclusive limit. Thus, if $N=10$, *! N, N−3 . . . 0* would execute for $N=10, 7, 4,$ and 1. This simple construct includes all the popular loops but never leaves a doubt about its exact meaning.

Similarly, *? E* encompasses *if/then/else*, as in

```
? P< 10: ....
   =@: ....
```

as well as the case construct:

```
? L= 'A': ....
   = 'B': ....
   = 'C': ....
   = @: .....
```

```
for i := 1 to m do
begin
  c := 1;
  for j :=1 to m do
    if A(i) > A(j) then c := c + 1;
  A(c) :=: A(i)      (* interchange A(c) and A(i) *)
end
```

Listing 1.

POWER! SPEED! VERSATILITY!
Brady Book Tools

Put You on the Cutting Edge of Computer Language Technology!

UNIX and XENIX: A Step-By-Step Guide *Douglas W. Topham and Hai Van Truong*

Get your copy of this main selection of the Small Computer Book Club (March '85)! Written for those using 16 bit microcomputers, this unique step-by-step guide gives you everything from the basic set-up of UNIX to advanced techniques for customizing the UNIX system. Covers UNIX commands, the C Shell, the Visual Editor, text-processing tools, and the Bourne Shell!
1985/508pp/paper/D9187-8/$21.95

The C Programming Tutor *Leon A. Wortman and Thomas O. Sidebottom*

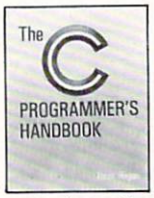
An introductory, "learn by doing" book that explores the inner workings of the C language and lets you create compact, transportable, fast-running programs quickly. You'll work with an abundance of programming examples and learn how to write useful, complex, multi-function applications programs as well as simple and valuable utilities.
1984/274pp/paper/D3642-8/$19.95

Your Books Can Be Ordinary
Or They Can Be Brady.

BRADY

Brady Communications Co., Inc.
A Prentice-Hall Publishing Company
Bowie, Maryland 20715

Toll-FREE (800) 638-0220 In MD (301) 262-6300



The C Programmer's Handbook *Thom Hogan*

Now you can readily find clear, complete information on all the important features of the C language! This definitive desktop reference is your aid in day-to-day programming and covers a wealth of information about C compilers, including documentation, examples, restrictions and defaults. Gives you to-the-point information about compiler-specific functions, operating system requirements, and much more!
1984/288pp/paper/D3650-1/$19.95

Mastering FORTH *Anita Anderson and Martin Tracy, MicroMotion*

Here's a step-by-step tutorial to this powerful, high-level, stack-oriented computer language! Gives you instruction on each of the commands required by the FORTH-83 International Standard, plus utilities and extensions that can be written within the Standard. Complete discussions on stack manipulation, variables, loops, strings, compiling words, and more! **1984/216pp/paper/D6609-4/$17.95**

8086/88 Assembly Language Programming *Leo J. Scanlon*

Now you can develop assembly language programs for any microcomputer using an 8086 or 8088 microprocessor! With this comprehensive text, you get a complete discussion of assemblers in general, plus a complete introduction to ASM-86, instructions for the 8086/88 assemblers, a concise introduction to the 8087 Numeric Data Processor, plus many useful programs, tables, and illustrations.
1984/213pp/paper/D424X-5/$15.95

ORDER TODAY!

YES!

I want to be on the cutting edge of computer language technology. Please send me the following maximum-performance book tools:

☐ **UNIX and XENIX: A Step-by-Step Guide** D9187-8/$21.95
☐ **The C Programmer's Handbook** D3650-1/$19.95
☐ **The C Programming Tutor** D3642-8/$19.95
☐ **Mastering FORTH** D6609-4/$17.95
☐ **8086/88 Assembly Language Programming** D424X-5/$15.95

Please indicate payment option:

☐ My check or money order is enclosed. Do not charge me for postage and handling. I have added state sales tax.
☐ Bill me. I agree to honor the invoice for the above checked products, which includes postage, handling and state sales tax.
☐ VISA ☐ MasterCard I agree to pay postage, handling, and state sales tax.

Card Number ______ Expiration Date______
Signature ______
NAME ______
ADDRESS ______
CITY ______ STATE______ ZIP______
Dept. JY

BRADY Brady Communications Co., Inc. • *A Prentice-Hall Publishing Company* • Bowie, MD 20715
Toll-FREE (800) 638-0220 In MD (301) 262-6300

The lack of key words permits total freedom in choosing names for variables and functions; it also helps the compiler. By setting a declaration like *ABC: @*, the function call simply degenerates into a label.

The rest of the notation and the basic math functions are commonplace, so the reader should have no problem translating the following example into any other language:

```
N: -9 . . . +9
X: -99.99 . . . 99.99
Y: @.@
IN: @
OUT:< - 'Y EXP N =',Y;
(< - 'ENTER X,N AFTER > >'
IN: < - '> >';
-> X,N;
Y := 1.0
   (? N = 0: OUT)
   (! N
      Y := Y * X)
   (? N<0:
      Y := 1/Y)
OUT;
IN)
```

My apologies for not having an ASCII keyboard available at this time; it makes the code look a little rough. But the idea does come across, I hope.

Dr. Max Schindler
Boonton, N.J.

COMPUTER LANGUAGE

Language dreams

Dear Editor:

I would like to point out another minor slight of APL, this time by John Snyder in "Recursive Procedures." (*COMPUTER LANGUAGE*, Apr. 1985, pp.19-24.) Recursion has been an important feature of APL right from the start, and I would certainly call APL one of the popular languages.

Having programmed extensively in FORTRAN, APL, PL/I, and C, my two favorite languages are APL and C. I hope your new language project seriously considers including the best features of these two languages.

It has interested me to discover the number of features APL and C already have in common. Those that come to mind readily are short names, recursion, embedded assignment, passing argument by value, modularity and extendibility through user-defined functions, processing at the bit level, funny symbols, a high degree of portability, zero origin for

CIRCLE 20 ON READER SERVICE CARD

indexing (optional in APL), the same syntax for labels, double floating arithmetic, automatic conversion of types as necessary to perform arithmetic, rowwise storage of arrays, simplicity of I/O without JCL, and continuing evolution.

While C is a hodgepodge, APL has an elegant simplicity and consistency of syntax that should be studied as a model for any new language. By having the same simple syntax rules for both user-defined and primitive functions, as well as straightforward right-to-left processing, there are no ambiguities or complicated tables of rules that need to be memorized or kept handy for reference. To me, this is what any new language should strive for while still trying to attain the versatility and richness found in both languages. The convenience of an interactive programming and debugging environment is also important.

From C should come the features that make it compilable without being overbearing and its structured flow control. We might as well use C's funny symbols and eliminate one of APL's handicaps also. Beyond this I haven't thought it through to its logical conclusions, so haven't anticipated what problems might be encountered, but I can dream, can't I?

Bob Stephan
Monterey, Calif.

BASIC benchmarks

BASIC benchmark tests (Tables 1 and 2) were inadvertently omitted from last month's BASIC wrap-up. We're sorry for any frustration that may have occurred due to this error. —Ed.

Benchmark tests for BASIC interpreters (min: sec)

Manufacturer and Product	Sieve test	Mysort	Matrix	Trig test sin ( )	Trig test cos ( )
MS-DOS interpreters					
American Planning Corp. MEGABASIC	14:10	1:22	0:49	4:53	3:54
Control-C BASIC Interpreter	error	7:58	6:03	—	—
IBM/Microsoft BASICA	36:30	2:57	2:06	3:32	4:17
Microsoft MS-BASIC 5.28	39:49	3:07	2:04	3:21	4:05
Morgan Computing Professional BASIC	18:45	2:03	2:21 1:40*	12:56 0:36*	12:57 0:36*
Ryan-McFarland RM-BASIC	13:54	1:17	1:52	10:45	10:57
Southwest Data Pluto BASIC	error	error	error	5:12	5:12
Summit Software Better BASIC	6:40	0:44	0:55	5:31	5:33
TransERA TBASIC	32:00	3:00	2:25	2:28	2:28
True BASIC	4:21	0:55	0:39	1:31	1:49
WATCOM BASIC	55:58	4:26	2:27 2:16*	5:17 1:42*	5:19 1:42*
CP/M 86 interpreter					
Digital Research Personal BASIC	27:28	2:19	3:02	2:52	2:49
CP/M interpreters					
DeltaSoft DeltaBASIC	error	12:27	8:02	8:43	—
Micro Mike's BaZic	5:03/i	4:23	2:30	8:43	9:11
Spectrum Logic DBASIC	4:37/i	3:41	3:32	3:56	4:01
APPLE MACINTOSH					
Apple MacBASIC	5:15	0:32	0:46	5:10	5:10
Microsoft MS-BASIC 1.0	24:13	1:47	0:67	4:51	4:51
Microsoft MS-BASIC 2.0b	21:53	1:49	0:70	1:10	1:10
Microsoft MS-BASIC 2.0d	22:10	1:47	0:73	4:52	4:52

*With 8087. i/=One iteration.

Table 1.

Benchmark tests for BASIC compilers (min: sec)

Manufacturer and Product	Sieve test			Mysort			Matrix			Trig test			
	Compile + link time	Exec file size (bytes)	Run time	Compile + link time	Exec file size (bytes)	Run time	Compile + link time	Exec file size (bytes)	Run time	Compile + link time	Exec file size (bytes)	Run time sin( )	Run time cos( )
MS-DOS compilers													
Digital Research CBASIC	0:35	9,216	0:16	0:33	9,216	0:02	0:45	9,728	0:20	0:39	12,800	22:10	22:12
IBM/Microsoft IBM BASIC Compiler	0:09	1,536	0:15	0:10	1,536	0:02	0:11	2,048	0:10	0:09	1,664	0:34	0:35
Microsoft Business BASIC Compiler	0:10	1,664	0:16	0:10	1,664	0:02	0:13	2,048	0:27	0:10	1,792	6:45	6:47
Microsoft BASIC Compiler	0:09	1,526	0:16	0:09	1,536	0:02	0:12	1,920	0:11	0:09	1,664	0:34	0:35
MicroWay 87BASIC	0:39	17,664	0:16	0:39	17,792	0:02	0:41	18,176	0:04	0:42	18,816	0:11	0:11
Softaid MTBASIC	0:04	15,182	1:24	0:04	15,182	0:09	0:07	15,182	0:27	0:05	15,182	5:37	6:02
Sparry BASIC B	—	—	—	—	—	—	<0:01	NA	1:09	—	—	—	—
CP/M compilers													
Alcor Systems Multi-BASIC	1:25	24,704	1:08	1:25	22,016	0:08	1:34	25,344	0:47	1:27	21,632	12:37	12:46
System/z BASIC/Z	0:55	24,704	2:02/i	0:55	24,704	2:12	0:60	25,344	2:22	0:66	24,407	2:31/i	2:32/i

i/ = One iteration. NA = Not applicable.

Table 2.

C Programmers:

Consider 104 Ways To Be More Productive

If you find and choose the right development software, you can: cut development effort, make impractical projects feasible, and eliminate unproductive, frustrating aspects of programming.

Confused? We'll help you sort thru the huge number of alternatives. Call for comparisons or information.

We carry 27 C Compilers, 4 C Intepreters, 49 Support Libraries, 5 C source debuggers, and 19 other C Add-ons for programming with MSDOS, Macintosh, or CP/M – more than 104 products, really. Here are some of the best products available:

Learn C Programming Only $95

"Introducing C" Interpreter

Computer Innovations has done it again! This interactive implementation is combined with a full screen editor and a thorough, self-paced manual.

You can develop programs faster by getting immediate feedback. Programs will start instantly upon your command. There is no need to wait for "compile and link."

Introducing C includes demo programs, powerful C language interpreter, complete C function library, full screen editor, color graphics, and C language compatibility. PCDOS $95

Simplify Screen Management

Windows for C

Keep your software up to date with the latest screen management features:

- Pop up menus and help files
- Instant screen changes
- Multiple windows
- Complete color control

Windows for C offers all of these plus much more in an integrated, compact, easy-to-use library of object code functions. Thorough, reference manual. Support for all memory models of popular C compilers. New version 3.1 offers enhanced portability and TopView compatibility. Full source available. MSDOS $180

Why Lattice C?

Lattice C Compiler

Trade mags such as Byte and PC Tech have nearly outdone themselves in praising Lattice C's speed and compactness.

Lattice C is a full implementation of K&R. It is compatible with any 8086 or 8088 and now has a 286 compile option.

Seven different memory models enable you to select the appropriate combination of addressability & efficiency to suit a particular situation.

Other specs include automatic sensing of an 8087 chip; Fork function; and complete I/O routines.

The thorough manual even includes subjects like interface to assembly language and machine dependencies. MSDOS $call

Add Communications Features to Your Programs

Greenleaf Comm Library

Greenleaf now enables you to communicate with remote systems or databases with an asynchronous communications library for C.

Individual transmission and reception ring buffers combine with an interrupt driven system. This eliminates the extra function of separately calling up the communications program.

Included are 3 library/object files, 220 functions; 230 page manual, complete source code, library tailor-made to suit compiler and memory, Hayes-compatible modem commands, and a complete sample file transfer program. MSDOS $169

Which Compiler Features Do You Need?

Optimizing C86 Compiler

Over the years the Optimizing C86 has evolved to be the most complete set of C compiler tools. It includes utilities, a rich library, and thorough tech support. In line 8087/287 routines run up to 100 times faster than the 8086 math package. The source code to all routines is included, so you have complete control over how they work. Thorough ROM support, Intel UDI & VMS cross versions are available.

More of the features you want include:

- special IBM-PC library • 2 math and 2 I/O libraries
- full memory utilization of the 8086/88/186/286
- compatibility with most commercial libraries
- object and source module librarian MSDOS $339

Fast File Access with Source

C-Index +

C-Index + contains a high performance ISAM, balanced B + Tree indexing system with *source* and *variable length* fields. The result is a complete data storage system to eliminate tedious programming and add efficient performance to your programs.

Features include random and sequential data access, virtual memory buffering, and multiple key indexes.

With *no royalties* for programs you distribute, full source code, and variable length fields C-Index + fits what you are likely to need.

Save time and enhance your programs with C–Index +. MSDOS $375

File Management: MultiUser/MultiLanguage

BTRIEVE

Billions and billions of bytes! That's what you can control with Btrieve's file manager. Btrieve gives you the ISAM capability you need without the maintenance headaches.

Using b-trees for optimum performance, Btrieve automatically maintains your files in sorted order on up to 24 different fields. And Btrieve offers you the fastest search algorithm available, to give you instantaneous access to any individual record. You can locate any record in 4 disk reads or less (thanks to Btrieve's RAM cache, usually less). With Btrieve you can stop wasting your time being a file clerk and concentrate on more productive tasks.

Btrieve's other features include:

- 4 gigabyte file size
- 4090 byte record length
- 255 byte key length
- duplicate, modifiable, and null keys
- up to 24 key indexes per file
- automatic file recovery after power failure

Btrieve's Local Area Network version lets you migrate your software to multiuser environments without changing your code. And offers you multiuser update capability beyond simple file locking schemes. Available *for all programming languages* as well as C. MSDOS.Single user $245. Multiuser $595.

Btrieve. Don't settle for less.

Call for details, comparisons, or for our "C Extras Packet" with over 50 pages of information about C support products.

THE PROGRAMMER'S SHOP

The programmer's complete source for software, services and answers

128- LC Rockland Street, Hanover, MA 02339 (617) 826-7531 (800) 421-8006

Ask about COD and PO's. All formats available. Prices subject to change. Names of products and companies are generally their trademarks.

CIRCLE 40 ON READER SERVICE CARD

CROSS X THOUGHTS

Binary trees and B-trees

By Namir Clement Shammas

In the last two columns we discussed hash-based searching for data resident in either RAM or storage disks. In this issue we will examine alternative techniques, namely, those based on tree data structures.

Efficient search methods that do not employ hashing must rely on having the data set up in some orderly fashion. For example, seeking specific information in a list of items is less time consuming if the list is sorted. Otherwise the entire list must be examined.

A sorted list can be maintained by performing an insertion sort with each new element added. A secondary index list can be used to speed up the search. It becomes apparent that as the list size grows, the average number of shifted list members increases.

Another way for maintaining data in sorted order, without the disadvantage of moving a lot of information, is to use binary trees. Trees are nonlinear data structures where each element, called a node, may be connected to two or more other nodes. A binary tree is one whose nodes are connected to two other nodes at most. Thus it is a two-way tree having a left and right pointer.

The topmost node is also called the root. This node is the first one added to the tree. The nodes that point nowhere (have null pointers) are called terminal nodes or leaves.

How is data insertion carried out in a binary tree? The very first inserted element becomes the root. Every other data element is inserted following a recursive algorithm.

Step 1. The root is the first node used in the comparison step.

Step 2. The search key of the new data element is compared with the node key. If the search key is less than the node key, the node's left subtree branch is searched. If not, the right subtree branch is searched.

Step 3. If either of the subtree branches is empty, then it becomes the location of the new key. The insertion process terminates. Otherwise it continues at the next step.

Step 4. The first node encountered in the left or right branches becomes the new comparison node. The process is resumed at step 2.

Searching in a binary tree is conducted similarly. The same comparison scheme is used until either a match to the search key is found somewhere or the search fails at a terminal node.

Deleting a node from a binary tree is not a simple operation. The type of node deleted (root, nonterminal or terminal) decides the complexity involved. Terminal nodes are the easiest to delete. Other nodes will leave behind gaps to be filled, which is done by reorganizing the portion of the tree that was connected to the deleted node.

The worst case is when the root is deleted. The right subtree nodes are rearranged. The whole process looks like a power struggle in a dicatorship. When a death occurs in the government, the lower ranked the deceased, the less disturbance there is in the land. The reverse is also true, especially when the "head" (the root) of the country passes away. In that case there is a power struggle among the next-in-rank players and their clans (corresponding to subtrees).

Binary trees allow for efficient searching and maintaining sorted lists kept in computer memory. In languages like Pascal and Modula-2, nodes can be dynamically allocated and/or deallocated, creating trees in memory whose sizes are limited by hardware capacity. Pointers are used to access nodes.

Dynamic trees spare the programmer from being locked in a predefined, fixed size array representing the nodes. Recursive search methods allow for easier algorithms for the traversal of the tree. Terminal nodes would cause an automatic backtracking to higher nodes. All these advantages are lost when fixed arrays using nonrecursive searches are employed. Now the terminal nodes must be supplied with special pointers to guide the search resumption. This kind of binary tree is called threaded.

Binary trees may be used to store keys in memory (when a data maintainance session is in progress) while the complete records are stored on disk. This allows for a faster RAM-based search and one-disk access. This scheme is feasible in the case of sufficient memory, and the search keys represent a small portion of the data records.

Maintaining a balanced binary tree is important for fast searching. The insertion and deletion of keys occur in an unpredictable order. This can very well lead to unevenly distributed branches. The worst case is to have all the nodes in one subtree while the other is empty. This creates a big linked list! It happens when a perfectly sorted array is inserted in a binary tree.

The AVL trees (named in honor of two Soviet mathematicians, G.M. Adelson-Velskii and E.M. Landis, who described the balancing method) use algorithms that keep the binary trees in tune. This is done first by assigning a balance factor to each node, reflecting the tilt in its subtrees. There are three cases: tilted to the left, right, or balanced.

As new nodes are added and subtrees grow, there is a tendency to widen the difference in the heights of different subtrees. The insertion algorithm incorporates code lines that deal with any imbalance as soon as it occurs (Listing 1). The action taken is to rotate subtrees either to the left or right to restore the balance. As PPL code shows, there is a lot of node rearrangement done. Deleting nodes from an AVL tree requires moving nodes around and adjusting subtrees to maintain the balance.

Since the AVL binary trees maintain balanced subtrees, one can use the following technique to increase the search speed in a large AVL tree. The method will actually cause the tree structure, by academic definition, to cease being a true binary tree.

We can attach "zoom" pointers along the outer node edges in both left and right subtrees, starting with the root. Each zoom pointer establishes a double-link between nodes and another that is *n* nodes downward. The search begins with using the zoom pointers, which allows for faster traversal along the edges. Every time a key comparison test fails, we use the next zoom pointer to locate the next node.

DeSmet C

8086/8088 Development Package $109

FULL DEVELOPMENT PACKAGE
- Full K&R C Compiler
- Assembler, Linker & Librarian
- Full-Screen Editor
- Execution Profiler
- Complete STDIO Library (>120 Func)

Automatic DOS 1.X/2.X SUPPORT

BOTH 8087 AND S/W FLOATING POINT

OVERLAYS

OUTSTANDING PERFORMANCE
- First and Second in AUG '83 BYTE benchmarks

SYMBOLIC DEBUGGER $50

- Examine & change variables by name using C expressions
- Flip between debug and display screen
- Display C source during execution
- Set multiple breakpoints by function or line number

DOS LINK SUPPORT $35

- Uses DOS .OBJ Format
- LINKs with DOS ASM
- Uses Lattice® naming conventions

Check: ☐ Dev. Pkg (109)
☐ Debugger (50)
☐ DOS Link Supt. (35)

SHIP TO: ____________

____

____________ ZIP____

CWARE CORPORATION

P.O. BOX C
Sunnyvale, CA 94087
(408) 720-9696

All orders shipped UPS surface on IBM format disks. Shipping included in price. California residents add sales tax. Canada shipping add $5, elsewhere add $15. Checks must be on US Bank and in US Dollars. Call 9 a.m. – 1 p.m. to CHARGE by VISA/MC/AMEX.

Street Address: 505 W. Olive, #767, (94086)

CIRCLE 11 ON READER SERVICE CARD

When the key comparison test fails or we reach the terminal nodes, we retreat to the last node searched. We resume searching by examining neighboring nodes.

A variation of this scheme is to collect the zoom pointers in two separate lists or arrays, one for each primary subtree. A comparison with the root key decides which list to use. Instead of dealing with zoom pointers sequentially, we use the interval halving technique. We first compare the search key with the node key of the median zoom pointer. This decides which half of the list will be used in the next search. We select the median zoom pointer of the new half-list and so on.

What about searching in data files? How applicable are binary tree techniques? The answer is that basically there is nothing wrong with applying the preceding to data files, except for the slowness of the hardware. Accessing data from a disk is much slower than accessing information from computer memory.

The solution is to read as many keys at a time and use them in multiway decision making. Notice that we said keys and not records, because the former are usually much smaller than the latter. This helps in having more dense information per unit storage.

Immediately we realize that we have expanded the concept of a binary or two-way tree into an m-way tree. Instead of accessing single nodes from RAM, we read pages or buckets of keys or records. Such trees are called B-trees (Figure 1) after R. Bayer.

Like a binary tree, a B-tree has a root page or node (containing multiple keys). The number of keys per page is also called the order of the B-tree. The keys are arranged by insertion sorting to maintain the advantage of searching a sorted list. Each key has a page pointer and a data record pointer associated with it. To resume searching, the page pointer is used to point to the next page. When a key matches a sought key, the data record pointer is used to access the entire record from the data file. Terminal pages or nodes are called leaves. Their data structure is identical to any other nodes.

B-trees, by definition, must follow certain structural rules and behavior. While this may seem an imposed restriction, it actually gives the B-tree a sustained balance and fast search capability.

- All leaves are on the same level
- All nodes and leaves are at least half full.

Donald Knuth has suggested that each page be two-thirds full. This type of tree is called the B* tree.

Listing 2 shows the PPL code for search routines in B-trees. The next CrossThoughts column, which resumes the modified B-tree discussion, will present the PPL code for insertion. It is very similar to that of B-trees.

The code presented is nonrecursive. The search starts at the root page. Procedure *SearchNode* is used to compare the search key with the page keys. If a match is found, the search in the B-tree ends, and the appropriate data record pointer is used to access the information sought from the data records file.

Conversely, the comparison yields the pointer for the next page to be searched. This takes us to the next page located one level lower than the current one. This process repeats itself until the leaf pages are searched.

A search to insert keys in the B-tree is similar. The existence of matching keys may flag an error if stored keys are to be unique.

How do B-trees grow? How are keys rearranged in the process? The answer lies in the fact

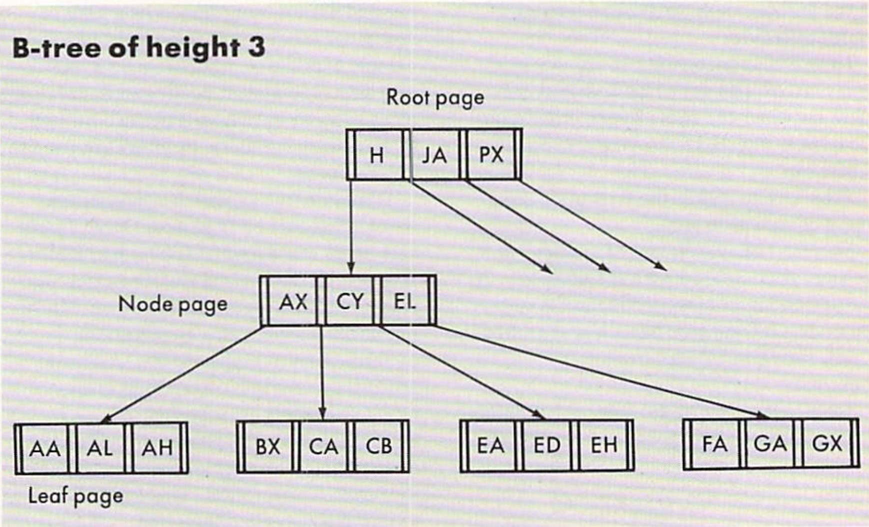


Figure 1.

that their growth is very much affected by the structural rules mentioned earlier.

Initially there is one empty page. As keys are added and fill out the page, the following takes place when attempting to add a key to a full page. The page keys read in memory and the new key is inserted in the key list such that a perfect sorted order is maintained. Next, the median key is selected, dividing the list into two halves. The lower half is written back to the old page, while the upper half is written to a newly created page. The median itself is stored in another, higher level, new page.

This makes the B-tree grow by one level. The page containing the median key becomes the root page. A comparison with the latter key will guide the search toward either leaf page.

This scenario takes place at the first growth stages of the B-tree. As more keys are added, they are all inserted in the leaf pages. As each leaf page becomes full, it is split into halves and the median key inserted into the parent node page. If the latter becomes full, the same operation is carried out, resulting in two new, half-full nodes. Their median is inserted or used to create a new parent page node. The cascading effect is very evident when adding a new key to a B-tree filled at a certain height. This yields a new root page and an increase in tree height.

Deleting keys from B-trees dictates that the remaining structure not violate the definition rules. If this leaves a page less than half full, one of two things may occur. If an adjacent page is more than half full, it will export a key to the key-deficient page. Otherwise the two pages are combined. This may affect the parent nodes and require additional adjustments.

While B-trees are very advantageous to use, they certainly have their drawbacks. One of them is to search for a key and then read subsequent or previous keys, producing an ascending/descending sorted list. This process involves a lot of jumping between pages with a relatively high number of I/O accesses.

Is there a solution to this problem? Yes—it is called the B+ tree, the subject of out next column. We will also discuss an additional modification for the B+ tree.

Binary trees have applications in subjects other than sorting and searching. They can be used as parsing trees reflecting the order and priority of code execution, such as mathematical expressions. This application will be discussed in a future column.

I look forward to hearing from you if you have any comments or thoughts to share about this column's subject.

```
AVL binary tree insertion routines

---------------------------------------------------------- AVL tree

EXPLICITLY RECURSIVE PROCEDURE Insert(VAR Root: Node_Pointer;
                                      New_Rec : Node_Rec;
                                 VAR IsTaller : BOOLEAN)


-- Inserting a new node in an AVL tree and keeping balanced tree
-- HasTallerSubtree is of type BOOLEAN
-- Balance_Factor is [-1..1] reflects the tilt of the tree

IF Root = nil
THEN Set Root to point to New_Rec
     BalFactor = 0; IsTaller = TRUE
     New_Rec.Left_Ptr = nil; New_Rec.Right_Ptr = nil

ELSE
    IF New_Rec.Key <= Root^.Key
    THEN
        Insert(Root^.Left_Ptr,New_Rec,HasTallerSubtree)
        IF HasTallerSubtree
        THEN
            CASE BalFactor OF
                WHEN -1 => LeftBalance
                WHEN  0 => BalFactor = -1; IsTaller = TRUE
                WHEN  1 => BalFactor =  0; IsTaller = FALSE
            END CASE
        ELSE IsTaller = FALSE
        END IF
    ELSE
       Insert(Root^.Right_Ptr,New_Rec,HasTallerSubtree)
        IF HasTallerSubtree
        THEN
            CASE BalFactor OF
```

Listing 1 *(Continued on following page).*

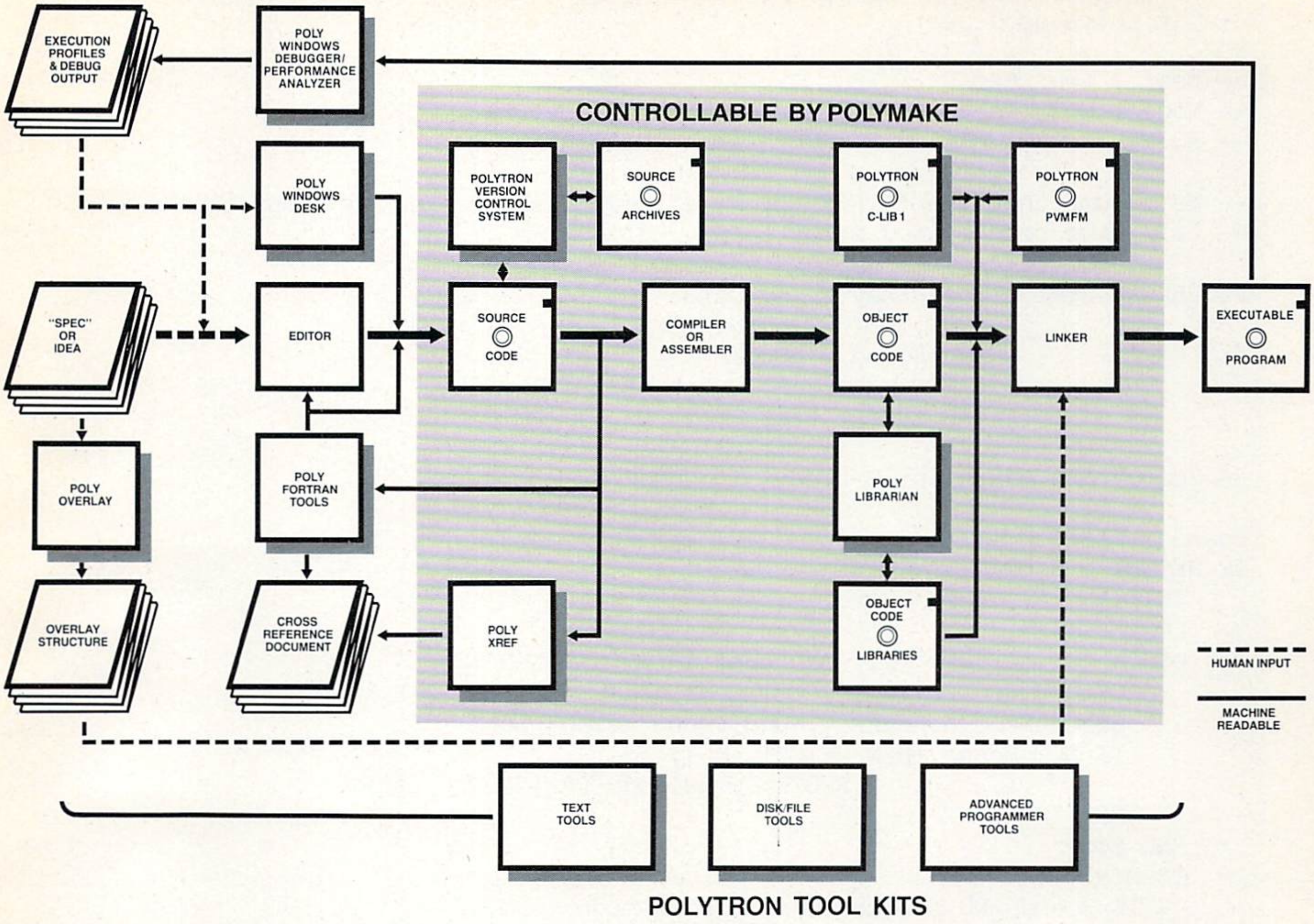
```
                    WHEN -1 => BalFactor = 0; IsTaller = FALSE
                    WHEN  0 => BalFactor = 1; IsTaller = TRUE
                    WHEN  1 => RightBalance
                END CASE
            ELSE IsTaller = FALSE
        END IF
    END IF
END IF
END Insert


PROCEDURE RightBalance
-- P1 and P2 are of type Node_Pointer

P1 = Root^.Right_Ptr
CASE P1^.BalFactor OF
    WHEN 1 => Root^.BalFactor = 0; P1^.BalFactor = 0;
              RotateLeft(Root); IsTaller = FALSE
    WHEN -1 => P2 = P1^.Left_Ptr
               CASE P2^.BalFactor OF
                 WHEN  0 => Root^.BalFactor =  0; P1^.BalFactor = 0
                 WHEN -1 => Root^.BalFactor =  0; P1^.BalFactor = 1
                 WHEN  1 => Root^.BalFactor = -1; P1^.BalFactor = 0
               END CASE
               P2^.BalFactor = 0; RotateRight(P1); RotateLeft(Root)
               IsTaller = FALSE
    WHEN OTHERWISE => -- do nothing
END CASE
RightBalance


PROCEDURE RotateLeft(VAR Ptr : Node_Pointer)

-- Tempo_Ptr is a local Node_Pointer

IF (Ptr = nil) OR (Ptr^.Right_Ptr = nil)
THEN DISPLAY "Error"
ELSE Tempo_Ptr = Ptr^.Right_Ptr; Ptr^.Right_Ptr = Tempo_Ptr^.Left_Ptr
     Tempo_Ptr^.Left_Ptr = Ptr; Ptr = Tempo_Ptr
END IF
END RotateLeft
```

Listing 1 *(Continued from preceding page).*

```
B-tree searching routines

---------------------------------------------------------- B-tree

-- Data types involved are:

-- Node_Rec = RECORD
--      Count_Node_Key : INTEGER
--      Node_Key : ARRAY[1..MAX] OF Key_Data
--      Node_Ptr, Data_Rec_Ptr : ARRAY[1..MAX] OF INTEGER
-- END RECORD
```

Listing 2 *(Continued on a following page).*

POLYTRON TOOLS IN THE SOFTWARE DEVELOPMENT PROCESS



Thousands of Professional Software Developers with demanding deadlines turn to high performance MS-DOS/PC-DOS tools from POLYTRON to boost their productivity. All the tools shown above are explained in detail in the **POLYTRON Programmer's Catalog.**

Network & Site Licenses Available.

PolyWindows Desk
The Modular, Expandable Desktop Organizer

100% memory-resident desktop organizer for professionals. Configurable so you can have one or more of the following functions available instantly: Hex/Decimal/Binary Calculator, Standard Calculator, ASCII Table, Multiple Document Text Editor, PolyKey for creating your own keyboard macros, Rolodex-type files, Calendar, Alarm Clock, Appointment Books, Autodialer, Grabber for cut-and-paste between applications. **Copy-Protected $49.95, Not Copy-Protected $84.95.**

PolyWindows Debugger/ Performance Analyzer
A Professional, Memory-Resident Software Development Tool

Instantly analyze and debug ANY program running on your PC anytime. Superior to the IBM Professional Debugger and easier to use. Supports multiple breakpoints, multiple display formats, memory searches, single-step execution, disassembly, hex arithmetic, and block moves. Tells you where your program is spending time and allows you to optimize performance. You can specify the memory ranges for analysis. Results are displayed graphically or in tables. **Includes PolyWindows Desk (Not Copy-Protected) all for only $149.** (Present PolyWindows users can receive trade-in discounts.)

POLYTRON Version Control System (PVCS)

Efficiently maintains revision history of source files for software project management. Maintains chronological, historical records of changes as "edit scripts" (reverse deltas) with full text of the latest "checked in" revision. Reconstructs any prior "revision" of any module, defines a "version" as specified revisions of various modules, supports branching from prior revisions, optional password protection. Friendly, unobtrusive user interface is based on a unique "syntax guide" concept with context-sensitive help. Designed for single or multiple programmer projects. A powerful tool with mainframe power. **Single-user license: $395.**

PolyXREF
The Multi-Lingual Cross Reference System

Generates Cross Reference listings across single or multiple source files of one or more languages. Tells you exactly where each variable and procedure is defined and everywhere it is referenced. **Package includes PolyXREF engine plus one source language module plus .CRF module: $129, additional source language modules [C, Pascal, Assembler or English]: $49 each. Complete package including all four language modules: $219.**

PolyLibrarian
The Object Module Library Manager

The premier library manager allows you to create, examine and manage libraries. Modules can be listed, added, replaced, extracted, and deleted. You can search for public or external names and even change them. Three operating modes can be intermixed for maximum flexibility. **PolyLibrarian I for Microsoft format libraries: $99, PolyLibrarian II for Microsoft and Intel format libraries: $149.**

PolyMake
The Intelligent Program Builder and Maintenance Tool

Automates the software maintenance process. Determines which files (programs, etc.) are out of date and invokes your compiler, linker, librarian and does whatever is necessary to bring your entire system up to date. Frees you from the need to remember which files depend on others and which files have been modified. Remembers the exact sequence of operations necessary to create a new revision. Completely automatic unlike competing products. Once you use it, you can't live without it. **Only $99.**

POLYTRON C Library 1
Over 65 High Performance Routines For Lattice C Compiler Users

These are routines for serious programmers that need Executive and I/O functions. Complete source code (mostly Assembler) included. **Only $99.**

POLYTRON Virtual Memory File Manager (PVMFM)

Provides efficient virtual file access and buffering for both text and binary files with automatic buffer-to-disk swapping. Reduces program access time and handles data structures larger than memory allows. PVMFM is a library of user-callable functions that may be linked with your applications code to provide virtual file management capability and random access file buffering. Compatible with Assembler and high-level languages. **Only $199.**

To Order Products Call
1-800-547-4000
Ask for Dept. No. 350
Foreign & Oregon orders call
(503) 684-3000
Send Checks, P.O.s To:
POLYTRON Corporation
P.O. Box 787 DS-350
Hillsboro, OR 97123
Add $5.00 Shipping To Total Order

To Order detailed Programmer's Catalog, send requests to address above for fast response (or circle reader service card number).

POLYTRON
High Quality Software Since 1982

CIRCLE 34 ON READER SERVICE CARD

```
-- Variables used are:

-- Leaf : Leaf_Rec
-- Node : Node_Rec
-- ROOT, HEIGHT, MAX, MIN, NUM_PAGE : INTEGER

-- We assume that the "BTREE_Key_File" key file and "Record_File" data
-- file have been opened prior to any call.

PROCEDURE Search(Soughtkey : Key_Data
                 ROOT, HEIGHT : INTEGER
                 VAR Found : BOOLEAN
                 VAR SoughtLeaf, Sought_Loc : INTEGER
                 VAR Leaf : Leaf_Rec)

-- Search procedure for B-tree

Found = FALSE
IF HEIGHT > 0 THEN
    INITIALIZE: None
    LOOP
    BEGIN IF (HEIGHT <= 1) OR Found THEN EXIT END IF
        READ BTREE_Key_File,ROOT,Node
        SearchNode(SoughtKey,Node,Found,Sought_Loc)
        IF NOT Found THEN HEIGHT -= 1;
                          ROOT = Node.Node_Ptr[Sought_Loc]
        END IF
    END LOOP
    TERMINATE:
       IF Found AND search is in access mode THEN
           READ Record_File,Node.Data_Rec_Ptr[Sought_Loc],Records
       END IF
END IF
END Search

-------------------------------------------------------------------
PROCEDURE SearchNode(SoughtKey : Key_Data;
                     Node : Node_Rec;
                     VAR Found : BOOLEAN;
                     VAR Sought_Loc : INTEGER)

BEGIN

    IF SoughtKey < Node.Node_Key[1]
    THEN
      Found = FALSE
      Sought_Loc = 1

    ELSE
     INTIALIZE: Sought_Loc = Node.Count_Node_Key
     LOOP
     BEGIN IF (SoughtKey >= Node.Node_Key[Sought_Loc]) OR
              (Sought_Loc <= 1) THEN EXIT END IF
           Sought_Loc -= 1
     END LOOP
     TERMINATE: Found = (SoughtKey = Node.Node_Key[Sought_Loc])
    END IF
END SearchNode
```

Listing 2 *(Continued from a preceding page).*

Lifeboat.™

C is the language. Lifeboat™ is the source.

Productivity Tools from the Leading Publisher of C Programs.™

The Lattice® C Compiler

The cornerstone of a program is its compiler; it can make the difference between a good program and a great one. The Lattice C compiler features:

- Full compatibility with Kernighan and Ritchie's standards
- Four memory model options for control and versatility
- Automatic sensing and use of the 8087 math chip
- Choose from the widest selection of add-on options
- Renowned for speed and code quality
- Superior quality documentation

"Lattice C produces remarkable code. . .the documentation sets such a high standard that others don't even come close. . .in the top category for its quick compilation and execution time and consistent reliability."

Byte Magazine

Lattice Library source code also available.

Language Utilities

Pfix 86/Pfix 86 Plus — dynamic and symbolic debuggers respectively, these provide multiple-window debugging with breakpointing capability.
Plink 86 — a two-pass overlay linkage editor that helps solve memory problems.
Text Management Utilities — includes GREP (searches files for patterns), DIFF (differential text file comparator), and more.
LMK (UNIX "make") — automates the construction of large multi-module products.
Curses — lets you write programs with full screen output transportable among all UNIX, XENIX and PC-DOS systems without changing your source code.
BASTOC — translates MBASIC or CBASIC source code directly to Lattice C source code.
C Cross Reference Generator — examines your C source modules and produces a listing of each symbol and where it is referenced.

Editors

Pmate — a customizable full screen text editor featuring its own powerful macro command language.
ES/P for C — C program entry with automatic syntax checking and formatting.
VEDIT — an easy-to-use word processor for use with V-PRINT.
V-PRINT — a print formatting companion for VEDIT.
CVUE — a full-screen editor that offers an easy way to use command structure.
EMACS — a full screen multi window text editor.
Fast/C — speeds up the cycle of edit-compile-debug-edit-recompile.

Graphics and Screen Design

HALO — one of the industry's standard graphics development packages. Over 150 graphics commands including line, arc, box, circle and ellipse primitives. The **10 Fontpack** is also available.
Panel — a screen formatter and data entry aid.
Lattice Window — a library of subroutines allowing design of windows.

Functions

C-Food Smorgasbord — a tasty selection of utility functions for Lattice C programmers; includes a binary coded decimal arithmetic package, level 0 I/O functions, a Terminal Independence Package, and more.
Float-87 — supports the 8087 math chip to boost the speed of floating-point calculations.
The Greenleaf Functions — a comprehensive library of over 200 routines.
The Greenleaf Comm Library — an easy-to-use asynchronous communications library.
C Power Packs — sets of functions useful for a wide variety of applications.
BASIC C — This library is a simple bridge from IBM BASIC to C.

Database Record Managers

Phact — a database record manager library of C language functions, used in the creation and manipulation of large and small databases.
Btrieve — a sophisticated file management system designed for developing applications under PC-DOS. Data can be instantly retrieved by key value.
FABS — a Fast Access Btree Structure function library designed for rapid, keyed access to data files using multipath structures.
Autosort — a fast sort/merge utility.
Lattice dB-C ISAM — a library of C functions that enables you to create and access dBase format database files.

Cross-Compilers

For programmers active in both micro and mini environments we provide advanced cross-compilers which product Intel 8086 object modules. All were developed to be as functional — and reliable — as the native compilers. They are available for the following systems:

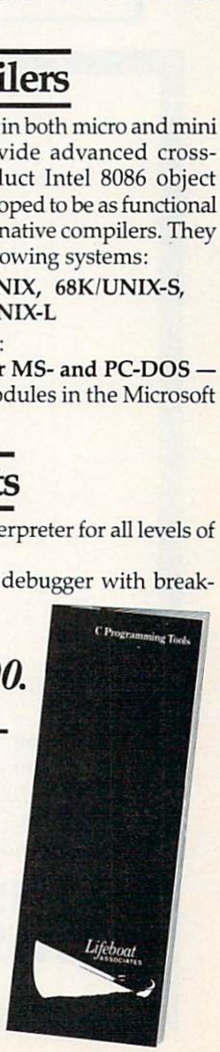
VAX/VMS, VAX/UNIX, 68K/UNIX-S, 68K/UNIX-L

Also, we have available:
Z80 Cross-Compiler for MS- and PC-DOS — produces Z80 object modules in the Microsoft relocatable format.

New Products

Run/C — finally, a C interpreter for all levels of C Programmers.
C Sprite — a symbolic debugger with breakpoint capability.

Call LIFEBOAT: 1-800-847-7078. In NY, 1-212-860-0300.



YES! Please rush me the latest FREE Lifeboat™ catalog of C products.

*Name*____________________ *Title*__________
*Company Name*________________ *Business Phone*__________
*Address*______________________________
Please check one of the following categories:
☐ *Dealer/Distributor* ☐ *End User* ☐ *Other*__________

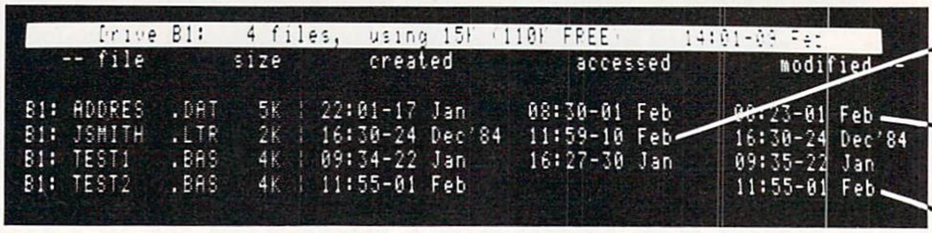
Return Coupon to: Lifeboat™ Associates
1651 Third Avenue, New York, NY 10128

© 1985 Lifeboat Associates

CL

CIRCLE 85 ON READER SERVICE CARD

DATESTAMPER™ has the answers



```
  Drive B1:    4 files,  using 15K (110K FREE)     14:01-09 Feb
  -- file        size       created              accessed             modified --

B1: ADDRES   .DAT    5K | 22:01-17 Jan      08:30-01 Feb      08:23-01 Feb
B1: JSMITH   .LTR    2K | 16:30-24 Dec'84   11:59-10 Feb      16:30-24 Dec'84
B1: TEST1    .BAS    4K | 09:34-22 Jan      16:27-30 Jan      09:35-22 Jan
B1: TEST2    .BAS    4K | 11:55-01 Feb                        11:55-01 Feb
```

When did we print that letter?

Has the mailing list been updated?

Which is the latest version?

DateStamper™ keeps your CP/M computer up-to-date!

- avoid erasing the wrong file
- back-up files by date and time
- keep dated tax records of computer use
- simplify disk housekeeping chores

OPERATION: DateStamper extends CP/M 2.2 to automatically record the date and time a file is created, read or modified. DateStamper reads the exact time from the real-time clock, if you have one; otherwise, it records the order in which you use files. Disks initialized for datestamping are fully compatible with standard CP/M.

INSTALLATION: Default (relative-clock) mode is automatic. Configurable for any real-time clock, with pre-assembled code supplied for popular models. Loads automatically at power-on. **UTILITIES:** • Enhanced SuperDirectory • Powerful, all-function DATSWEEP file-management program with date and time tagging • Disk-initializer • Installation and configuration utilities **PERFORMANCE:** Automatic. Efficient. Invisible. Compatible.

Requires CP/M 2.2. Uses less than 1K memory. Real-time clock is optional.

When ordering please specify format

8" SSSD, Kaypro, Osborne Formats **$49**

For other formats (sorry, no 96 TPI) add $5.

Shipping and handling **$3**

California residents add **6%** sales tax

MasterCard and Visa accepted

Specialized versions of this and other software available for the Kaypro.
CP/M is a registered trademark of Digital Research, Inc.

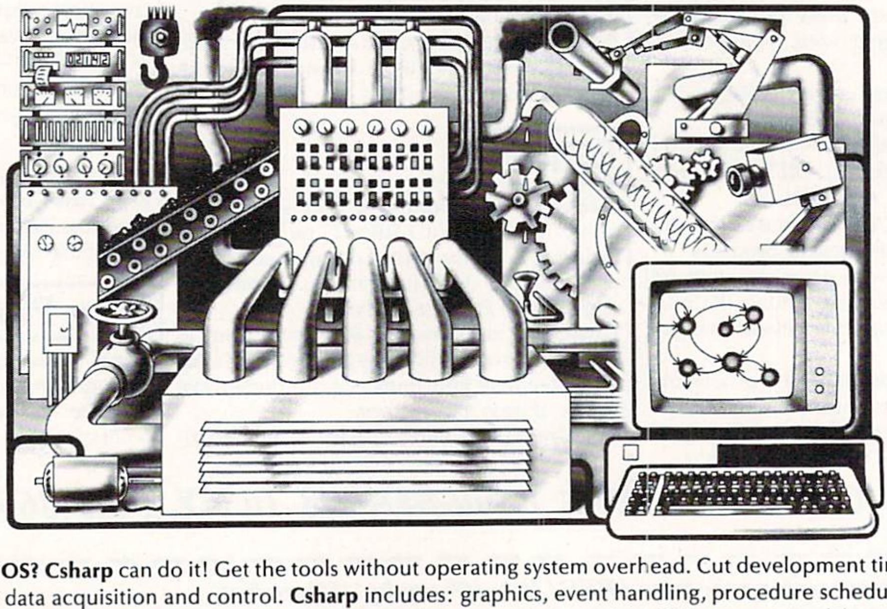
Write or call for further information

Plu*Perfect Systems

BOX 1494 • IDYLLWILD, CA 92349 • 714-659-4432

CIRCLE 35 ON READER SERVICE CARD

Csharp Realtime Toolkit



Realtime on MSDOS? Csharp can do it! Get the tools without operating system overhead. Cut development time with C source code for realtime data acquisition and control. **Csharp** includes: graphics, event handling, procedure scheduling, state system control, and interrupt handling. Processor, device, and operating system independent. **Csharp** runs standalone or with: MSDOS, PCDOS, or RT11. **Csharp** runs on: PDP-11 and IBM PC. **Csharp** includes drivers for Hercules and IBM graphics boards, Data Translation and Metrabyte IO boards, real time clock, and more. Inquire for Victor 9000, Unix, and other systems. Price: $600

SYSTEMS GUILD

Systems Guild, Inc., P.O. Box 1085, Cambridge, MA 02142
(617) 451-8479

CIRCLE 27 ON READER SERVICE CARD

DESIGNERS DEBATE

The dilemma of copy protection

By Ken Takara

In the days before personal computers, when mainframes ruled the world, software was not bought and sold as it is today. A company would license its program for use by a client company to run on its computer. Naturally, with the cost of computers, a company typically had only one system, usually rented from a hardware vendor. Such things as software piracy, copy protection, and shrink-wrap packaging were unknown.

The computer industry of today, however, has an entirely different appearance. A site may have a dozen or more computers in one place. Software may be found retailing for $100 or less. And software piracy is a common concern.

This month, Designers Debate presents an argument clinic focusing on the general area of copy protection, licensing, and the responsibilities of software producers and users. The list of participants includes: Allen Amaro of Parsec Research, a systems and software house; Ward Christensen, prolific writer of public domain software; Pete Rowe, independent consultant and former director of R&D at The Learning Company; John Kennedy, a long-time programmer from the mainframe days; Dennis Hamilton, software designer; Dave Pifer, founder of Computer Hobbyists Against Raiders and Thieves (CHART), an organization devoted to countering computer crime by educating the public; Jim Kyle, system operator of the *COMPUTER LANGUAGE* forum on CompuServe; Ben Sevier; Jeff Brenton; Scott Sharkey; Gerald Edgar; Tony Gambacurta; Todd M. Roy; and Joseph P. Salemi.

Let's start by talking about copy protection—what it is and what it does.

Pifer: It doesn't. And the entire idea is counterproductive.

One of the more annoying aspects of copy protection is that it gets good, honest users thinking along different lines. Given a protected program, a basically honest person's thoughts soon turn to cracking the code.

Sure, they do it for legitimate purposes, but the dissemination of the cracking info leads to more and more complications. And, of course, the professional pirates just go about their merry ways. So the people being affected are the normal folk.

Sevier: I deal with the business end user daily. I refuse to let *any* of my clients buy protected software if there is any other possible way to accomplish the job. My clients are all very small businesses and can't afford to be without back-up protection. I am also strongly urging hard disk systems for all my accounting system clients. A system that can't be run without a floppy or key in place is next to useless.

Brenton: I have decided to return a $1,600 package of programs today that, while not copy protected, has a license agreement that says I can only make three copies. I normally keep two copies on hard disk, three or four tapes in the archive, plus one set of floppy backups, for a total of 10 to 14 backups. And if I don't send in the signed shrink-wrap license agreement (the terms of which were hidden by the substantial box) saying that I agree with everything they say, they will provide no support!

This is just before a page of the usual "We don't even guarantee that it will work!" To read the license terms, I had to break the only seal on the package, which means, in their opinion, that I already agree to their terms. And this company expects me to trust my business's accounting future to them. Fat chance!

Christensen: Not only do I not buy copy protected software, I don't buy other products from companies that protect their software. I am a happy owner of SuperCalc III. Lotus can keep their copy protection. I never bought DBase-xxx even though I had a $200 chance to upgrade from my CP/M version—it's copy protected.

I am an honest person. I know there are dishonest people in the world, but I don't think software should be protected.

Sharkey: Now you really asked for it! Software companies are treating illegal copying as if it were the disease itself rather than a symptom of the disease. What disease? The software companies' disease.

There is still a lot of trash software being sold—often with significant bugs—which is useless yet still expensive. There is no legal recourse for a buyer who gets stuck with one of these lemons as most software warranties aren't worth the paper they're printed on.

Software is overpriced, regardless of what the manufacturers say. Look at the typical explanations most companies give for high prices. First, there is "perceived value." What a crock!

If the perceived value of Lotus 1-2-3 is $495 because of all the time it saves you, then why do people copy it? The fact is that the great majority of buyers do not perceive the value to be $495. Only Lotus does.

Next, there is the "high cost of support." Another crock! Have you ever tried to get support from one of these companies? If you go to the dealer you are just wasting your time, as he either doesn't know anything or will just try to sell you another package. If you call the company, you are put on instant half-hour hold at your own expense because "all our lines are busy." And then, this support is usually useless anyway.

Kyle: Great response! Not all software publishers fail to back their products, though. I can name three (all small time) that provide total support, and one of these gives a written guarantee that the product will do exactly what the manual says it will do or be repaired or replaced at the publisher's expense. If only the Big Guys would do the same, most of the problems would begin to fade away.

Most folk are willing to pay for good support if they get it. By restricting that support to registered owners, piracy becomes unattractive.

Kennedy: The practical problem for large vendors is consequential damages. Little firms don't have to be afraid, realistically, of being sued for millions of dollars because of a failure in somebody's payroll. Big firms do, and their lawyers would be insane to let them give a guarantee.

Did you ever write a totally bug-free program longer than 25 lines or so?

RUN/C:™
Finally,
a C Interpreter

Available NOW for only **$149.95!**

Finally, a painless introduction to the C language. With **RUN/C: The C Interpreter** you can create and run C language programs in an environment as easy to use as BASIC.

RUN/C is C for the rest of us. It is a robust implementation of standard K&R. **RUN/C** is for both the beginner and professional.

RUN/C includes full floating point, 8087 support, structures, unions, casts and more than 100 built-in C functions.

With **RUN/C** you get all this with a command structure modeled after BASIC's using familiar terms such as EDIT, RUN, LIST, LOAD, SAVE, TRON, SYSTEM, etc.

Since **RUN/C** is a true interpreter it means that C programs can be written, tested and run within a single protected environment. It is a teaching tool and a source code debugger.

Here's more good news . . .

- Great documentation: a 400-page, easy-to-read manual filled with executable programs
- Array-index and pointer bounds checking
- Variable-trace and dump diagnostics PLUS an integral program profiler
- Full buffered and unbuffered file I/O
- Printer and asynch support
- Forking to your favorite full screen editor with automatic return to **RUN/C** with your edited program
- System Requirements: IBM® PC or compatible with PC-DOS 2.0 or MS™-DOS 2.0 or greater with ANSI.SYS.

Get things right the first time with **RUN/C: The C Interpreter.**™

For immediate delivery or more information, call:

1-800-847-7078
(in N.Y. **1-212-860-0300**)

or write: Lifeboat Associates™
1651 Third Avenue
New York, NY 10128

RUN/C is a trademark of Age of Reason Co.

CIRCLE 87 ON READER SERVICE CARD

Kyle: I only guarantee that the program will do what I say it will in the manual. There is an explicit exclusion of consequential damages! Despite extensive tests by people who usually find most bugs, I still have to make good on it from time to time.

All I ask is that the Biggies do exactly the same thing: guarantee that it does what they say when installed per directions and exclude all claims of consequential damages, or for that matter, fitness for purpose. There's no way to tell what a buyer expects, but it's easy to control your own performance claims.

Kennedy: Another problem is that large programs such as compilers or word processors of the sophistication of Microsoft Word are difficult to define. Just when is something a bug? For example, Microsoft is getting a lot of static just now because the Macintosh version of Word does weird things with very long footnotes.

But what, precisely, should a general purpose word processor do with a footnote that is several pages long? The situation gets even nastier with AI. How do you know for sure when you have a bug in a game from Infocom? One of their recent games, in fact, has something that looks just like a bug. Early on in the game, the display of the "time" in the story, shown at the top of the screen, stops changing.

People think they've found a bug until they discover that they can prevent it from happening by saying "WIND THE WATCH." I take Infocom as an example because it's a clear example of software that does more than the author himself originally imagined.

In the case of a square root routine, for example, you can say that the routine works if, for any number: 1) it extracts the square root to the maximum precision possible given the word length of the result and the original imprecision due to the word length of the argument, or else 2) it correctly says that the square root cannot be taken because the argument has none.

So that's easy. But in a natural language input situation with only partially quantifiable data, many of the possible input interactions may never have been actually defined in any strict sense. How can you be sure that a bug is there?

Amaro: We've just announced an upgrade to our product for owners of the previous package. I've been getting calls at the rate of about 10 per day. And more than half of them can't give me the serial number. I got a call from one person for a replacement manual because he "spilled coffee on it." I even got a request for a refund. And none of these people could give any proof they had purchased it.

Pifer: The best protection you have is to keep diligent records of those people who have bought authorized copies. Then the coffee spills, lost serial numbers, and so on aren't such a problem. You know who owns your product and the threat of rip-off is not so serious.

Kennedy: Just take a look over on GameSig on CompuServe. Fairly often someone will come up with how his mother threw away the Infotator from Sorceror or the chart from Starcross, both of which are necessary to get past randomly varying puzzles in the games. Standard operating procedure is to tell them, "Gee, I hope you sent in your registration. I'm sure Infocom will be willing to send you a new package."

Roy: I like the idea of one software developer. He encodes the serial number and the name of the purchaser in every program he sells. Every time the program is run, it comes up with the serial number and name.

Any attempt to remove or change the number or name causes the program to erase itself from memory as well as attempt to remove itself from the disk. Otherwise the user can make as many copies as he wishes. You would be less prone to giving it away knowing that your name will be presented every time it is used, even by someone else.

Amaro: Perhaps it's just the market we've been selling to—the mass market.

Pifer: It's a safe bet that mass market software is more prone to piracy than packages tailored to professionals. Of course, the primary reason for this is that the mass market has not been educated to the fact that piracy is theft. John and Jane Consumer have a very limited perception of what is involved in creating a viable software product and even less of an understanding of the repercussions of their acts.

Rowe: The probability of software copy protection being broken is directly proportional to its perceived value by the kids who break it. This means that something like VisiCalc, which has little value, isn't vulnerable. However, SkyFox—most of Electronic Arts' products for that matter—has a very high perceived value and is always being worked on.

Once a kid has broken the protection, he usually has a network of friends that he passes it to. After that, he may go to the Source and update the parameter list for one of the copiers, like LockSmith. Or he may start distributing it outside his own group.

Kennedy: Not just consumers. My company is putting in quite a few PCs. Only this morning, someone told me, "Look what I have! Someone made me a copy of this word processor so that I won't have to use EDLIN." I told her that she could be fired for it and pointed out that this was theft. She just told me not to worry and after all, it wasn't a new product, and in short, acted like a pre-Civil War slaveholder faced for the first time with an abolitionist.

It was as though I had somehow committed a monstrous breach of etiquette. And this is a mainframe system programmer in her 30s, not a particularly psychopathic personality, and certainly not ignorant of the labor that goes into any significant piece of programming.

Frankly, I don't think people are basically honest. Most people think of themselves that way, but the plain fact is that whenever we want to indulge ourselves in something a bit underhanded, we always manage to find a good explanation why *this* is a special case.

Does this hold out much of a promise for removal of copy protection? I doubt it.

Hamilton: Everything I've seen that permits arbitrary copying without allowing redistribution involves some sort of key-and-lock technique that ties the software to a given system. If the key is on a special disk, there is the problem of having it in the machine in order to work the program.

The alternative is some sort of key that is confirmed by inspection of the host computer. There are machines that have their serial numbers in machine-readable locations and that helps. Usually there needs to be a special technique for getting the first copy onto the system, but it's straightforward after that.

There is also a method in which a hardware key is plugged into the computer in order to run a program. Often it plugs into a game port on the side of the computer and is read by the program.

Roy: ADAPSO has been turned down, thank the great JuJu. Actually, hardware keys are nothing new. Years ago, someone made a graphics editor for the TRS-80 that required a key to be plugged into the cassette port. It went under.

Salemi: Besides, we all know how difficult it is to keep track of all the disks! Imagine trying to organize keys for all of them? Glad the idea went nowhere.

Hamilton: Copy protection should allow all useful application of the software by the purchaser. The idea is to force the thieves to demonstrate guilty knowledge that their actions are unlawful, not to make foolproof protection or bring harm to anyone's use of their computer.

This *does* discourage people. And those it doesn't hinder at least can't proclaim cow-eyed innocence that their actions are not reprehensible. Since I don't believe in restricting a purchaser's ability to make copies as part of the meaningful use of the software, I guess we need to distinguish between techniques that discourage piracy by preventing copying and by other means.

Pifer: The software industry would be a far sight better off to invest the money in education rather than wasting it reinventing the copy protection wheel. I'll stand by a common belief that most folks who know right from wrong will pick right. Those that would pick wrong are going to pick wrong no matter what.

Edgar: One of the excuses one sometimes hears is, "These copyright restrictions are not enforceable and probably not legal anyway, so I'm within my rights to copy this."

Kyle: I just saw in the paper that American Brands, the tobacco company, is defending itself against a piracy suit filed by MicroPro over WordStar with the claim that shrink-wrap licenses are simply not valid since there is no chance of negotiating an agreement.

A number of the attorneys in the software field have told me that they share this belief and would happily defend on such grounds.

The 1980 revision to section 117 of the 1976 federal copyright act reads, in part, "it is not an infringement for the owner of a copy of a computer program to make, or authorize the making of, another copy or adaptation of that computer program provided (1) that the new copy or adaptation is created as an essential step in the utilization of the computer program in conjunction with a machine and that it is used in no other manner, or (2) that the new copy or adaptation is for archival purposes only . . . "

Rowe: One thing you have to think about also is what do you do about prosecution? Can these small publishers get involved in prosecution? Without it, there's no bite to the protection or copyright.

Amaro: One of the things you have to do is aggressively go after pirates in order to maintain your copyright. Otherwise you can easily lose your rights to the software.

Kennedy: Someone in *PC Week*'s letters column just said that he wouldn't consider any software to be fairly priced until it reached the price of a hardcover book. So until it gets there, he'll just go on stealing it.

I'm really beginning to think that nothing but a few well-publicized jail terms will do any good. Don't these people have any idea how much labor goes into a 100,000-line program? Or of the difference in degree of detail work between something you hack together for your own use vs. a crash-proof, idiot-proof, all-possible-input-proof commercial product?

Salemi: Dave Pifer has the right idea. Education is the answer. It's nice to see that many corporations are making the effort to inform their employees of copyright laws and are setting company poli-

UNIX ® STYLE TEXT EDITOR

for Z80 ® with
CP/M2—CP/M80 ®
and compatible systems

use a UNIX editor on your micro !

Magnificent programers editor
Features powerful text manipulation

- fast, optimum use of memory (from 20K tpa)
- extended UNIX matches on search and swap:—
  ^ $. [set] [not set] * +
- string variables in swaps
- versatile character editing (open mode)
- macro command files
- conditional and interactive commands
- flexible file I/O includes user numbers
- files up to 20 000 lines, 4 million characters
- video or hardcopy terminal, no installation required
- 17 page reference manual, 58 page tutorial, *Help* command
- *Advanced* tutorial is value on its own.

Features:
  sorting: multiple & complex keys, such as dates and topological sorts
  form letter processing
  multi column tables
  case conversion

have the same editor on your mini and micro

Introductory price ONLY US$100

Also available:
  GREP — text file search, count, regular expressions
  ROFF—vanilla UNIX implementation, handles large documents

SPECIAL PRICE for
E, GREP, ROFF US$150

XYLOGIC
P.O. Box 1407
CHATSWOOD. 2067
AUSTRALIA

MasterCard VISA

Trademarks: UNIX Bell Labs,
CP/M2, CP/M80 Digital Research.
Z80 Zilog inc.

CIRCLE 28 ON READER SERVICE CARD

cies against copying for personal use. As long as they enforce the policy, they have done a major first step in the education process.

Software companies can do a great deal to help this along by providing more liberal site licenses. If a company can make the copies it needs under a license, then it sets the example for employees by obeying the law. And the software company makes money in the process. This route seems to have much greater potential for helping both the end users and the software company.

Rowe: I have experience working with schools. Face it, kids in the classroom are rough on diskettes. They need a way to make backups. In many cases, school districts have adopted the purchasing policy that if it doesn't have provision for backups, they will not purchase it.

There are several ways we can handle the case of schools. One is noncopy protection. Another is to permit multiple loads in the classroom without a local area network. Then there's the issue of licensing that the Minnesota Educational Computing Consortium is pushing.

The school market is a very large market. But they are typically on a very tight budget. They can't afford to buy much software at the current price. It's always been that a school would buy a computer, plug it in, turn it on, and say, "Hey! Well, now what?"

"Well, now you either have to create or buy some software!"

"Oh no! We ran out of money buying the computer!"

Surprise, surprise. They really don't have a lot of funds. If they have multiple machines in the classroom, these $40 or $50 computer products have got to be able to serve on all those machines.

Pifer: The little guys in this business have it tough enough without the added problems and expense of going the copy protected route. While I won't buy protected software from even the biggies, I most certainly would not buy a protected disk from some small guy who may not be in business six months from now when I need to buy a replacement disk.

Roy: I really think that pirates simply like the challenge of breaking protected software and then thumbing their noses at the producers. While we're on the subject, how many copy programs are themselves copy protected? (Answer: Most.) Isn't there a contradiction somewhere?

Rowe: It's like a phase of maturation for the hacker. Many of them go through it. Generally, I think, they are the high school kids who are bright enough not to have to spend all their time on their homework. I think the worst fear of a publisher is the boot trace, where you actually trace through the code, separating out the protection code. You can do it if you've got the time. It's all a matter of time. No sane publisher would consider himself permanently protected.

Gambacurta: There is an argument that suggests that software piracy helps the publishers to some degree. The idea is that the proliferation of a package gives it exposure. Giving a product exposure is important for market development. I don't personally buy this argument, but I think it is interesting.

Kennedy: There is also the approach of demo or sampler copies of software. This is a version of the program with some critical part—such as *SAVE* or *PRINT*—left out. For example, there was the free copy of Microsoft Word in the subscriber copy of PC World. I have also seen cheap, $9 versions for sale with a rebate coupon toward the real thing.

Pifer: As you know, I am the founder of CHART, located here in the Writers and Editors SIG of CompuServe, trying to bring matters of ethics, morals, and common sense into play in the computer field. If and only if education is given a fair shot and has proven ineffective will I buy the "bigger and better protection is our only hope" premise of most software companies.

Instant-C™:
The Best Value
In C Programming Tools

The edit-compile-link-test-debug cycle that takes tens of minutes with compilers and linkers is only seconds with the *Instant-C* interpreter. Yet it runs your programs **50 to 500 times faster** than conventional C interpreters! You get the best of both compilers **and** interpreters. Only *Instant-C* is a complete, integrated environment for creating, testing, and running your programs.

Instant-C gives you **all** of these proven capabilities in one tightly integrated package:

interpreter—*Instant-C* runs your programs faster than some compilers; has direct execution; full K&R

compiler—*Instant-C* can make stand-alone programs

full-screen language editor—shows syntax errors with cursor set to trouble spot

C source debugger—single-step, breakpoints, stack trace, more

run-time checker—validates pointer refs, array bounds, more

C source formatter—save editing time, find logical flaws

standard library with source—for best portability

linker—work with multiple source modules

Lint—extensive compile-time validation

The cheapest available examples of these tools would cost $800 (and they don't even work together). You could spend close to $3000 to get the best product of each kind, but you'd have ten times the complexity, filling megabytes of disk. *Instant-C* is faster: it performs these functions automatically. *Instant-C* is far more than the sum of its parts.

Instant-C is all of these capabilities in one package, fits on a single floppy disk, is full K&R, works on IBM PC's, compatibles, and others under DOS or CP/M-86. It costs only $495.

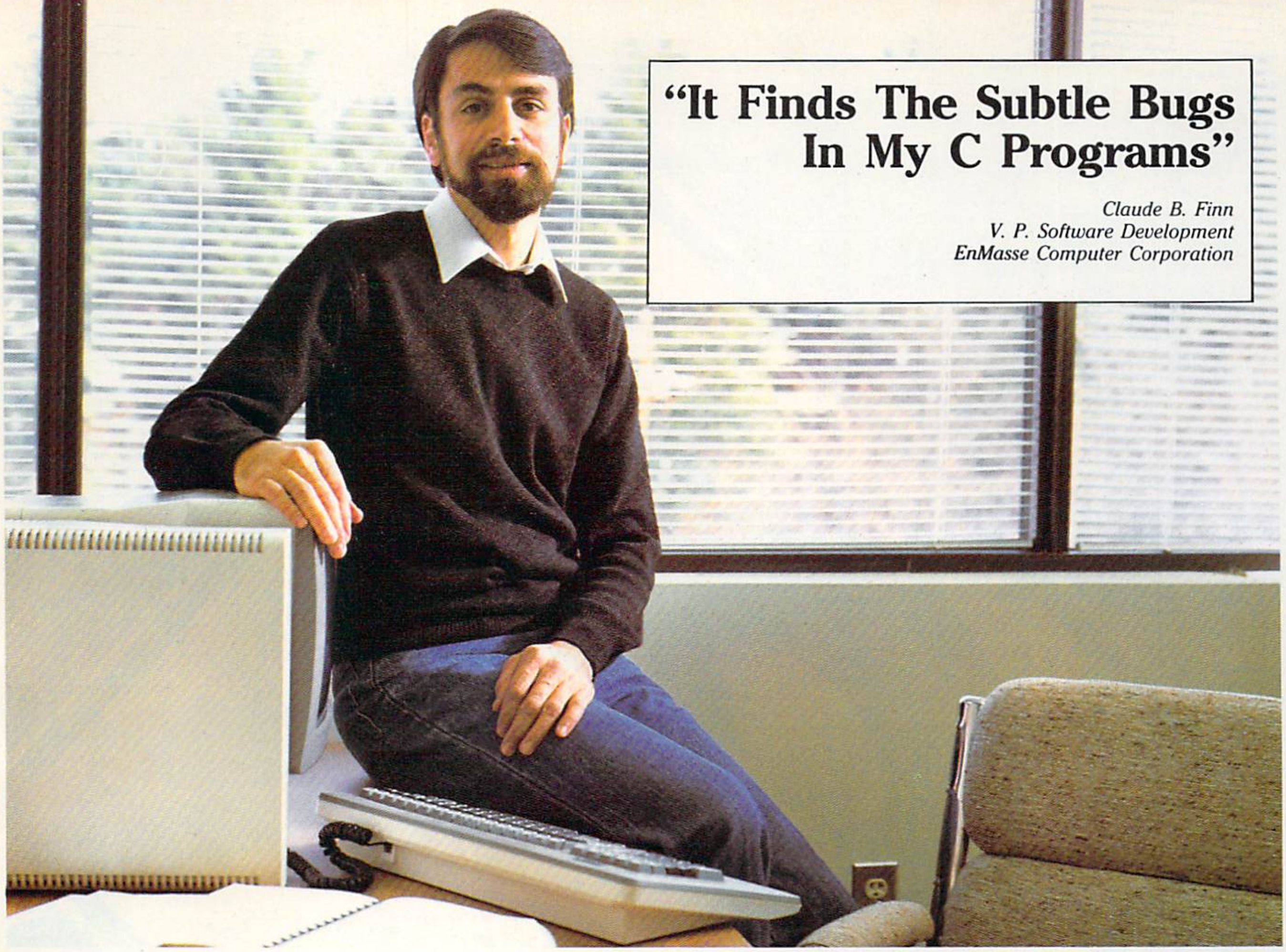
Instant-C is the best value in C programming tools. Guaranteed, or your money back for any reason in first 31 days.

Rational
Systems, Inc.

(617) 653-6194
P.O. Box 480
Natick, MA 01760

Instant-C is a trademark of Rational Systems, Inc.

CIRCLE 72 ON READER SERVICE CARD



"It Finds The Subtle Bugs In My C Programs"

Claude B. Finn
V. P. Software Development
EnMasse Computer Corporation

The SAFE C™ Family Can Literally Cut Software Development Time In Half. For UNIX™ and VAX/VMS.™

"Evasive bugs that use to eat up days — I'm finding them in minutes. Stray pointers, errant array indexes, parameter mismatches, misuse of string functions...I'm using Safe C automatic error detection to find them all."

Claude Finn is one of the many C programmers who have discovered that the Safe C family of software development tools dramatically enhances programmer productivity and improves software reliability and portability. Most Safe C customers have recouped their investment in these tools within the first month of active use. And with the security of Safe C their programmers are sleeping a lot easier!

The Safe C Family includes the Runtime Analyzer, Dynamic Profiler, Standalone Interpreter, English to C Translator and C to English Translator. They may be purchased separately or as a group.

CCA Uniworks, Inc.
Productivity Tools for Programmers
20 William Street • Wellesley, MA 02181

For more information or to place an order call our customer representatives at

800-222-0214

In MA (617) 235-2600
or mail this request form today.

Please send me information on:

☐ The Safe C Development Tool Family
☐ AI Development Tools
☐ CCA EMACS Unix/VMS Editor Environment
☐ Your complete line of state-of-the-art programming tools

☐ Please send license forms

Name ____________________
Title ____________________
Company ____________________
Address ____________________
City, State, Zip ____________________
Phone (   ) ____________________

CCA UNIWORKS, INC.
20 William Street • Wellesley, MA 02181

A Crowntek Company

Unix is a trademark of Bell Laboratories, VAX and VMS is a trademark of Digital Equipment Corporation, MS/PC-DOS is a trademark of Microsoft, Safe C is a trademark of Catalytix Corporation, CCA EMACS is a trademark of CCA Uniworks, Inc. CL685

CIRCLE 60 ON READER SERVICE CARD

Give your Forth the recursive powers of LISP and C

Forth: Twithe Curthed, too!

By Jean-Pierre Schachter

LISP and C programmers take special pride in their languages' compact code and often argue over which language's brevity-to-power ratio is more favorable. The method offered here for adding recursion to Forth implementations may give Forth the prize, considering the power recursion adds and that the code consists of one short line. Of course, LISP and C already support recursion; the point is that in Forth a very powerful programming strategy can be added by means of very terse code.

Forth is clearly extensible in the sense that new programs (or words, as they are called in Forth) can be added in compiled form to the interpreter. But the really dramatic extensibility of the language is not seen until one understands that it is actually possible to add completely new control structures as well.

Most implementations of Forth come with a fairly rich selection of standard control structures, and so unless an application cries out for a very specialized one, this feature can easily be overlooked. As it happens, recursion is neither highly dedicated nor usually supplied and thus makes a very attractive illustration of Forth's extensibility.

Recursion

Recursion is an alternative to iteration that is particularly attractive in cases where the original task can be broken down into a finite set of smaller, similar tasks. In effect, recursion allows a word to call itself on parameters that redefine the scope of the call. This kind of re-call is most interesting for words that are functions—words that return a result—since each function call can be made to depend on and wait for the succeeding call's return before completing its calculation.

In such a series of calls, one can distinguish a wind and an unwind phase, the former consisting of the series of calls, the latter of the sequential completions of the calls. Where the recursive word is a function, typically only parameters are passed to the next call during the wind and only returns from completed calculations during the unwind.

In Forth, however, a variation on this is possible in which values (not returns, though) are pushed onto the stack during the wind for popping off during the unwind. Since each call in the unwind phase must know where to find the value being passed to it, recursion is most at home in an environment that includes a stack. In the best of cases, variables need not be used at all.

To ensure an end to recursions, the code must include a terminating condition. Where the word is a function, a predefined value should be returned on reaching the terminating condition for use by the next waiting, incomplete invocation. It is not necessary, however, that a word be a function in order for it to benefit from recursion.

A word may be a procedure rather than a function, and as such, have as its objective certain side effects—for example, changes in the values of variables, system calls, screen displays, etc.—rather than the returning of a value. Even if a value is returned, the value may be of no interest to the waiting, incomplete calls. This use of recursion is called tail recursion by LISP programmers.

Of the examples supplied later, the Fibonacci calculation (Fib) uses recursion in the first way mentioned, passing each call's return as a parameter to the waiting call during the unwind; the factorial calculation (FAC) is like the special Forth case, pushing values onto the stack during wind for use during unwind; and Quicksort is like the third case, a procedure that shifts stored values in an array during each call.

Using SMUDGE

Since Forth provides such exceptional control of its stacks, it is a natural for the use of recursive algorithms—those that generate values during wind and those that produce returns during unwind—since both need a place to rest their output for pickup.

Fortunately, it is possible to add recursion very easily to implementations based on the Forth Interest Group (FIG) standard, since FIG vocabulary includes the little used word *SMUDGE.* Despite its modest associations, *SMUDGE* is virtually all of the recursion to be added. To understand this, we have to take a look at Forth's dictionary and the way in which it is used at compile time.

Forth words are used exactly the same way commands are used in an interpreted and interactive language such as BASIC.

Creating a new word amounts to adding a command to the interpreter, a process that involves compiling the source code for the command and having it added to the interpreter's dictionary of available routines.

The problem for recursion occurs during the compiling process since the Forth compiler checks each command used in the new source code to see if it is already known (present in the dictionary). Normally, the word that is in the process of being compiled is not yet officially in the dictionary. It isn't officially in until compilation has been completed successfully. This means that using the same command as the one being compiled in the latter's source code will generate an error-based break.

This problem can be resolved once we understand why the word being defined is not found. It is not found because a stop flag is set at the head of the new word's name field in the dictionary—a stop flag that is not reset until the new word is successfully compiled.

This flag, however, can be toggled by the *SMUDGE* command so that the word in the process of compilation can actually be discovered in the dictionary search. In order to facilitate this process, we can define a new operator using any unused character, in my case the BAR (|), for inclusion in recursive source code. The definition is:

```
: | SMUDGE ; immediate
```

The immediate is important since it will make the BAR operate during compile time when it is needed. With this operator available, let us see the BAR in operation in the recursive calculation of *n*! (Listing 1).

FAC takes one parameter off the stack. If the parameter equals 1, then it duplicates it and multiplies the two together, yielding 1 on the stack for print out. If the parameter is greater than 1, then it places it on the stack, duplicates it, places a number 1 less on the stack, and calls itself. The effect is to push a parameter onto the stack during each call of the wind. For example, 5 FAC will place 5 4 3 2 1 on the stack. When 1, the terminating condition, is reached, the appropriate number of multiplications that have been waiting in the wings unfold as follows: 5 4 3 2, 5 4 6, 5 24, 120 leaving the final product either on the stack or put to the screen.

The operation can best be understood by noting that no call to *TORIAL* is complete—that is, reaches "*"—until the terminating condition, parameter = 1, has been satisfied. Once the terminating condition has been satisfied, the remainder of each unfinished call to *TORIAL* unfolds, in this case "*".

One might also wish to test the BAR in terms of its speed. For this purpose, we have a very convenient comparison base in the times recently listed in *COMPUTER LANGUAGE* (Feb. 1985, p. 79) for 21 C compilers.

Fib (Listing 2) clocked in at 114 sec (on my Osborne 1b with Software Toolworks' implementation of a FIG version of Forth) for 25 Fib (46,368), a very good time although it is last of the five CP/M compilers for which values were given. However, it is last only by 1 sec against BD Software's compiler and 10 sec for The Code Works'.

In addition, two of the seven CP/M compilers had no times, one because it was still working after 5 min, the other because it would not compile. This makes 114 sec a good time, given the exceptionally friendly development and debugging environment available in Forth and generally absent in traditional compilers. Fib also demonstrates that the BAR works even when multiple recursions are used in the word.

```
: TORIAL dup 1 = if dup * else dup 1 - | TORIAL | * then ;
: FAC TORIAL . ;
```

Listing 1.

```
      ( FIBONACCI number calc using BAR-RECURSE )

      : ONACCI dup 1 = if drop 0
                       else
                         dup 2 =
                         if drop 1
                            else
                              dup 1 - | ONACCI |
                              swap 2 - | ONACCI |
                              +
                         then
                  then ;

          : FIB ONACCI U. ;
```

Listing 2.

Indirect recursion

The cases of recursion discussed so far were all cases of what is usually called direct recursion, in which a particular

```
( INDIRECT RECURSION with BAR )

  variable RCSAD

: TEST1 ." OK! " cr RCSAD @ execute ; ( RCSAD holds TEST2's  )
                                      ( CFA address          )
: TEST2 | [ ' TEST2 CFA RCSAD ! ] |
      TEST1 ;                         ( from | to | sets the )
                                      ( TEST2 CFA address    )
```

Listing 3.

```
( QUICKSORT using BAR-RECURSE )
( set up for 20 integers )

  variable XX 40 allot variable MIDDLE

: | smudge ; immediate  ( '|' allows recursive calls )

: ENT .clear 40 0 do I 2 / 1 + . ." Int ? : "
     query 1 word number drop XX I + ! cr 2 +loop ;
                        ( entry word )

: PUT 40 0 do cr I 2 / 1 + . ." : " XX I + @ . 2 +loop ;
                        ( prints array of ints to screen )

  10 load

( QUICKSORT scr #2 )
                                      ( the QUICKSORT          )
: K@ 2 * XX + @ ;  : K! 2 * XX + ! ;  ( implementation is due )
: MIDDLE@ over - 2 / + K@ MIDDLE ! ;  ( to Gary Nemeth of      )
: COMPARE K@ MIDDLE @ - ;             (    Cleveland, Ohio     )
: EXCHANGE 2dup K@ swap K@ rot K! swap K! ;
: 2OVER 6 pick 6 pick ;    : 2SWAP 4 roll 4 roll ;
: SORT 2dup > if drop drop else  ( lo hi ___ )
    2dup 2dup MIDDLE@
    begin swap begin dup compare 0< while 1+ repeat
          swap begin dup compare 0> while 1- repeat
      2dup > not if 2dup exchange 1 -1 D+ then
    2dup > until swap rot
  2OVER 2OVER - rot rot - < if 2SWAP then
  | SORT | | SORT | then;
```

Listing 4.

FOR TRS-80 MODELS 1, 3 & 4
IBM PC, XT, AND COMPAQ

COMMERCIAL SOFTWARE DEVELOPERS and INDIVIDUAL PROGRAMMERS

appreciate MMSFORTH for its:

- **Power**
- **Flexibility**
- **Compactness**
- **Development speed**
- **Execution speed**
- **Maintainability.**

When you want to create the ultimate:

- **Computer Language**
- **Application**
- **Operating System**
- **Utility,**

BUILD IT in

mmsFORTH

(Unless we have it ready for you now!)

Bulk Distribution Licensing @ $500 for 50 units, or as little as pennies each in large quantities. (Corporate Site License required.)

The total software environment for IBM PC, TRS-80 Model 1, 3, 4 and close friends.

- Personal License (required):

MMSFORTH System Disk (IBM PC)	**$249.95**
MMSFORTH System Disk (TRS-80 1, 3 or 4)	**129.95**

- Personal License (optional modules):

FORTHCOM communications module	**$ 39.95**
UTILITIES	**39.95**
GAMES	**39.95**
EXPERT-2 expert system	**69.95**
DATAHANDLER	**59.95**
DATAHANDLER-PLUS (PC only, 128K req.)	**99.95**
FORTHWRITE word processor	**175.00**

- Corporate Site License Extensions . . . . **from $1,000**
- Some recommended Forth books:

UNDERSTANDING FORTH (overview) . . .	**$ 2.95**
STARTING FORTH (programming)	**18.95**
THINKING FORTH (technique)	**15.95**
BEGINNING FORTH (re MMSFORTH) . . .	**16.95**

Shipping/handling & tax extra. No returns on software.

Ask your dealer to show you the world of MMSFORTH, or request our free brochure.

MILLER MICROCOMPUTER SERVICES
61 Lake Shore Road, Natick, MA 01760
(617) 653-6136

CIRCLE 37 ON READER SERVICE CARD

NGS FORTH

A FAST FORTH,
OPTIMIZED FOR THE IBM
PERSONAL COMPUTER AND
MS-DOS COMPATIBLES.

STANDARD FEATURES
INCLUDE:

- 79 STANDARD
- DIRECT I/O ACCESS
- FULL ACCESS TO MS-DOS FILES AND FUNCTIONS
- ENVIRONMENT SAVE & LOAD
- MULTI-SEGMENTED FOR LARGE APPLICATIONS
- EXTENDED ADDRESSING
- MEMORY ALLOCATION CONFIGURABLE ON-LINE
- AUTO LOAD SCREEN BOOT
- LINE & SCREEN EDITORS
- DECOMPILER AND DEBUGGING AIDS
- 8088 ASSEMBLER
- GRAPHICS & SOUND
- NGS ENHANCEMENTS
- DETAILED MANUAL
- INEXPENSIVE UPGRADES
- NGS USER NEWSLETTER

A COMPLETE FORTH
DEVELOPMENT SYSTEM.

PRICES START AT $70

NEW◆HP-150 & HP-110
VERSIONS AVAILABLE

NEXT GENERATION SYSTEMS
P.O.BOX 2987
SANTA CLARA, CA. 95055
(408) 241-5909

CIRCLE 61 ON READER SERVICE CARD

word calls itself from within its own compiled definition. It is also possible to set up indirect recursive control structures in which a word's definition includes a call to a later word which, in turn, calls the calling word. The BAR will also allow this structure but with some additional complications.

The BAR was able to operate relatively simply for direct recursions because the address needed already existed—it was just a question of allowing the compiler to find it. In the case of indirect recursion, the address needed is unknown during the compiling process and consequently cannot be found. The solution is to supply the address when it becomes available during the compilation of the called word. In order to do this, one of the two (or more) words must do its calling by means of a pointer, an address contained in a variable. The code in Listing 3 provides a model for the relationship.

To simplify the code and make its strategy clearer, no terminating condition has been included and the word will crash after a time. TEST1 simply prints out "OK!" and a carriage return and executes the code at the address stored in RCSAD. TEST2 begins by invoking the BAR in order to toggle the *SMUDGE* bit and allow itself to be discovered. It then uses the left bracket to switch into run time and the tick to get TEST2's PFA (parameter field address) onto the stack.

It seems fairly standard in both Forth-79 and FIG to have the tick return the PFA and *EXECUTE* take it as its parameter. My own Forth seems to want the CFA (code field address), hence the next invocation in TEST2 is CFA, whose return is then stored in RCSAD. We next return to compile time by means of the right bracket and once again toggle the *SMUDGE* bit, finally calling TEST1. In effect, the code between the BARs in TEST2 finds the CFA of TEST2 and stores it in RCSAD during the compiling of TEST2 so that TEST1 will know where to call TEST2 when it is run.

The next word is based on a version of Quicksort written by Gary Nemeth of Cleveland, Ohio (*COMPUTER LANGUAGE*, Dec. 1984, p.9). His version was set up for IBM PC display management and presupposed the availability of *2OVER, 2SWAP,* and *RECURSE.* I've replaced the display management setup with a straightforward array, defined *2OVER* and *2SWAP*, and replaced *RECURSE* with the defined BAR. The sort requires that the low number of the list to be sorted be entered first on the stack, then the high, followed by the call to *SORT*.

The code in Listing 4 includes an integer entry word called *ENT* and an output word called *PUT*; it is set up to sort 20 integers. A typical run would have you command *ENT*, enter 20 integers into the array, then 1 and 20, followed by *SORT*, and finally *PUT*, which prints the sorted list on your screen. This Quicksort has two benefits: it demonstrates a recursive word that is more procedural than functional and, in addition, is a useful kind of sort in general.

Thus ends our short excursion into the land of self-reference, the strange enchanted land of snakes that devour themselves and liars who proclaim that all of their kind are liars. What you have learned here will enable you to turn to all of your LISPing friends and tell them that your Forth ith twithe curthed, too!

Dr. Jean-Pierre Schachter is associate professor of philosophy and dean of arts and social science at Huron College, London, Ont., which is affiliated with the Univ. of Western Ontario.

Microprocessor programming made simple.

"Keep it simple" was the principle of the 14th Century English philosopher William of Occam and it has even more validity today. Faced with the problems of sophisticated computer systems, designers have found that ever more complex programming languages are further complicating their tasks. Until now.

Occam. Created for system design and implementation.

When we started designing our new VLSI family of 10-MIP transputers, we built on William's simple philosophy. To take advantage of the possibilities opened up by the transputer, we needed to create a language capable of properly addressing parallelism and multiprocessor systems.

With the ability to describe concurrency (whether timeshared or real) and to handle message-passing at the lowest level of the language, all aspects of a system can be described, designed and implemented in occam. From interrupt handling through signal processing to screen editors to artificial intelligence. And on.

But occam is not limited to our transputer family. It provides an efficient, responsive implementation language for systems built on today's microprocessors. It also opens up future possibilities with its performance-enhancing multiprocessor capabilities. And INMOS now offers a product to let you exploit occam's total capability in your system.

Simplify your job with the Occam Programming System.

The Occam Programming System (OPS) gives you the tools for complete VAX / VMS software development. This package includes an integrated editor / checker, an optimizing VAX compiler and full documentation. This gives you a supportive environment for the development of occam programs for execution on the VAX. Cross-compilers for 68000 and 8086-based systems will also be available.

What's more, the occam programs developed and proven on the OPS will give you a head start for work with the INMOS transputer. Extensions to the OPS will be available which will allow occam programs to run on the transputer.

And if you have a requirement to program the transputer in other popular high-level languages, other extensions will include compilers for C, Fortran, and Pascal.

Get started today.

Contact us for our information pack on occam, the Occam Programming System and the transputer. You'll be surprised how simple your life can be.

For quick response, call us at (303) 630-4000 or write:
Occam, P.O. Box 16000,
Colorado Springs, CO 80935.

inmos

inmos, and occam are trademarks of the Inmos Group of Companies

FORTRAN
COBOL
BASIC
APL
PL/I
C
PASCAL
ADA
occam

CIRCLE 99 ON READER SERVICE CARD

HCR/PASCAL, WIRTH ITS WEIGHT IN C

PASCAL Originally designed by Niklaus Wirth, is now available for a wide range of UNIX™ processors. HCR/PASCAL conforms closely to industry standards, passes all conformance tests in the PASCAL Validation Suite. Supports multiple module programs, a dynamic string package, and direct random file access.

C is the standard language of UNIX, HCR/PASCAL is written in C and translates PASCAL into C producing efficient optimized code. This approach allows direct interaction with the UNIX environment and offers a high degree of portability.

UNIX is a powerful yet flexible operating system environment. HCR/PASCAL is available today on a diverse range of UNIX hardware: AT&T 3B™ series, the NCR Tower,™ DEC PDP-11/VAX,™ and others. HCR has a growing line of UNIX software including business applications. We back up all our software with full support. To find out how we can put HCR/PASCAL, C, and UNIX together for you, call or write:

Human
Computing
Resources
Corporation

HCR

10 St. Mary Street, Toronto, Ontario, Canada M4Y 1P9 (416) 922-1937

See us at UniForum Booth #1441

UNIX is a trademark of Bell Laboratories. PDP-11, VAX, and DEC are trademarks of Digital Equipment Corporation. AT&T 3B is a trademark of American Telegraph & Telephone. NCR Tower is a trademark of NCR Corporation.

CIRCLE 93 ON READER SERVICE CARD

Porting the UNIX Utilities

By Alan Filipski

C is widely acknowledged to be one of the most portable of the ALGOL-like, block-structured programming languages. One of the reasons for its portability, paradoxically, is the relatively low system-level abilities of C. Almost any low-level manipulation can be implemented efficiently as a C function, so there is not a strong tendency to extend the language.

Another reason for the de facto standardization of C is that for a long time there was essentially one textbook on C and one large company promoting C. This is obviously changing, and we can only hope that the newly formed ANSI/IEEE efforts toward the standardization of C will be as successful.

UNIX system utilities are commonly dismissed as being source-code portable since they are written in C. As a general statement, this is true, but there are several small exceptions.

This article gives a brief discussion and categorization of the C language portability problems that I and a team of coworkers at Motorola Microsystems encountered when converting UNIX System V utilities from a VAX to a M68000-based computer. The hurdles we overcame may help the reader understand some of the fundamental technical issues regarding the C language and its portability between different machines.

Many of the problems in our project were actually caused by latent bugs that were invisible on the original host. A few were caused by bad coding practices. All could have been avoided.

The System V/68 port

Several of our team members had been involved with a port before, and the port leader was an experienced UNIX system guru, but for most of us (the author included), it was a new experience. What we lacked in experience, we made up for in commitment and enthusiasm and learned a great deal along the way during this project.

A UNIX system port project naturally divides into three major subtasks: kernel, utilities, and compilers (including libraries).

The kernel of the UNIX system comprises the initialization, swapping and scheduling, process management, input/output, and device driver functionalities of the system. This is about 20,000 lines of code and contains a number of hardware dependencies. About 10% of the kernel code is written in assembly language. The kernel excludes some functionality commonly considered to be part of an operating system. The command interpreter sh(1), for example, is not part of the kernel, but is counted as a utility.

The compiler task includes the porting of compilers, assembler, linker, language preprocessor, etc. The development of the assembler and C compiler and the creation of the libraries are the main issues here. The system call interface and a small amount of library code are in assembly language and need to be rewritten, although most of the library code is in C. In our case, we chose to port the Bell Labs SGS II compiler. This was a cross compiler that ran on the VAX and produced M68000 code. To port it, therefore, we did not have to change the code generation functionality, but we only had to get the compiler to run on the EXORmacs computer.

Most of the code of the UNIX system, however, is for utilities, such as awk(1), grep(1), lint(1), and many others, including the shell (sh(1)) itself. These are written entirely in C. The parenthesized number refers to the UNIX manual section in which the utility appears. The utilities with which we are concerned are either in sections 1, 1M, 1C, or 1G. These designations refer to ordinary, system administration, communications, and graphics utilities, respectively.

Although the utilities are sometimes dismissed as being completely portable, we found that there were several man-years of work to be done in porting them from a VAX to a M68000. By far most of this work was involved with rewriting makefiles, packaging, testing, and reworking utilities—such as sdb(1), which has substantial amounts of functional machine dependency.

System V/68 utilities

The VAX version of System V has 303 utilities supplied by Bell. Any enumeration is necessarily ambiguous. For example, are cp(1) and ln(1) separate utilities? They both represent the same entry point in the same executable file. (I say they are two different utilities.) Is the entire graphics subsystem one utility or 57?

(One.) Is each game (Mastermind, Hangman, etc.) a separate utility? (Yes.) Is the SCCS system one utility or 13? (13.)

I counted only those utilities represented by executable files in the directories /bin, /usr/bin, /etc, or /usr/games. The exact number is neither well-defined nor important, but it appears that System V contains about twice as many utilities as the UNIX Version 7 operating system.

Of these utilities, 33 were defined by us (with Bell's concurrence) as not to be ported. These included the vpm and rje families of utilities and the games for which we had no source (such as Adventure). The absolute debugger adb(1) was also not to be ported since the symbolic debugger sdb(1) supersedes it. Twenty more utilities were deemed to be optional and were to be included in a subsequent release. These included most of the games and the graphics utilities. The number of utilities we are talking about here is thus approximately 250.

These utilities represent 287,606 lines of C source code in 960 source files. Present to control their compilation are 120 makefiles.

Nonportability issues

At the outset of the project, the utilities group did not have a clear idea of what sorts of problems would prove most troublesome. We had heard about the hibyte-lobyte problem (discussed later), and we suspected that some utilities, such as sdb(1), were by their nature extremely machine dependent. We eventually got a document from Bell called the "Port Acceptance Criteria" (see references), which gave estimates of the portability of all source files in the system. This document gave us some rough indications of the trouble spots, but by no means did it indicate all the problems.

At this point we might have run wholesale compilations of utilities for the M68000 and just tested them on our target system. Unfortunately, we had neither cross compiler nor target kernel at that time, so we had to do as much work as possible on the host VAX.

The first exercise performed by the group was the lint(1)/grep(1) search. Lint(1) (a utility designed to flag suspicious C language constructs) was run with the option string *-abhuvxp* on all utility source code. This option string suppresses everything except possible portability problems. Even with this suppression, lint(1) generated an enormous number of complaints, mostly relating to pointer casts and other slightly tainted uses of pointers. Because of its overwhelming size, the lint(1) output was not useful to us and we disregarded it.

In concert with the lint(1) exercise, a grep(1) source code scan on the utilities was also done. We searched for all occurrences of the strings *word, byte,* and *union*. This search was done with a known problem in mind, the hibyte-lobyte problem, i.e., the known difference in significance ordering of bytes between the source and target machines.

Though we knew it would be a problem, we were unsure of all its possible manifestations. The hibyte-lobyte problem is a good example of the interaction of architecture and the porting process. Since most of us had never before worried about such DEC idiosyncrasies (it took several patient explanations by one of the more avuncular members of our team before we understood why unions, for example, could cause the problem while masks and shifts could not), this grep(1) search turned out to be useful. Although most of the findings were false alarms, such as the word *password* in the source, some real problems were discovered.

Following an initial analysis of the grep(1) data for hibyte-lobyte problems, it was noticed that many machine-dependent areas had already been isolated. These areas were bracketed by conditional compilation directives with code inclusion based on the target machine. The machines defined included: VAX 11/780, PDP-11, AT&T 3B20S, and the IBM 370. The presence of these conditional compilation directives in a source code file was taken as prima facie evidence that the code required inspection for portability problems.

The grep(1) exercise finally resulted in a table of rough portability estimates. For each file, presence of any of the following suspicious constructs was noted:

- References to /dev/mem or /dev/kmem. Such references meant that the utility depended upon the structure of memory, which differs somewhat from port to port because of memory management considerations.
- References to object file format. Such references were important to us because we were going to use the new Common Object File Format (COFF) for our object files, while the system we were porting from used this format's predecessor. The only difference between the two is that the new COFF uses a string table to store arbitrarily long identifiers in the object file; the old format had limited-length identifiers.
- Presence of machine conditional compilation directives. It seemed reasonable to us that places containing conditional compilation directives warranted inspection since others found machine dependencies there.
- Presence of the words *word* or *byte*. By searching for the words *word* or *byte* we were trying to find comments describing any byte-order dependent coding practices.
- Presence of the word *union*. As described later, unions provide a way for C code to be machine dependent.

At this point, we began examining by eye all source code indicated as suspicious and making any changes of which we were sure. (All our work was being done under Project SCCS control, of course. See *COMPUTER LANGUAGE* Nov. and Dec. 1984.) Eventually, as we got our compiler (a Bell SGS II cross compiler running on the VAX and generating code for the M68000) and as our target system became more stable, more and more testing was possible. The rest of this section is an itemization, in retrospect, of the C language portability problems we encountered in porting the utilities to the M68000. It should be a useful guide for anyone who needs to do similar work.

Hibyte-lobyte problem. One of the problems in some of the utility code involved the hibyte-lobyte issue. This is the classic UNIX system portability problem. It refers to the difference in significance ordering of bytes within short (16-bit) or long (32-bit) words.

On DEC equipment, the first (lowestaddressed) byte of a multibyte integer is the least significant, while on most other machines, including the M68000, the first byte is the most significant. In most ordinary C programming, this difference is of no consequence. It is possible, however, to write C code that has different effects in these two environments.

For example, if we declare the variable *x* as a *union*:

```
union { char c[4];
    int i; } x;
```

and execute the assignment *x.i=1*, then on the VAX, *x.c[0]* will be equal to 1 and *x.c[1]*, *x.c[2]*, and *x.c[3]* will be equal to zero. On the EXORmacs, however, *x.c[3]* will be equal to 1 and *x.c[0]*, *x.c[1]*, and *x.c[2]* will be equal to zero. In terms of bits, *x* looks like this on the VAX:

```
00000001
00000000
00000000
00000000
```

and like this on the M68000-based machine:

```
00000000
00000000
00000000
00000001
```

There are also a few other ways to invoke this machine dependency in C, such as writing data to a file in one binary format and reading it back in another. Shifts are not a problem. Regardless of the internal representation, compilers insure that, for example:

```
x >> 10
```

is, for any positive integer *x*, the same as

```
x/1024
```

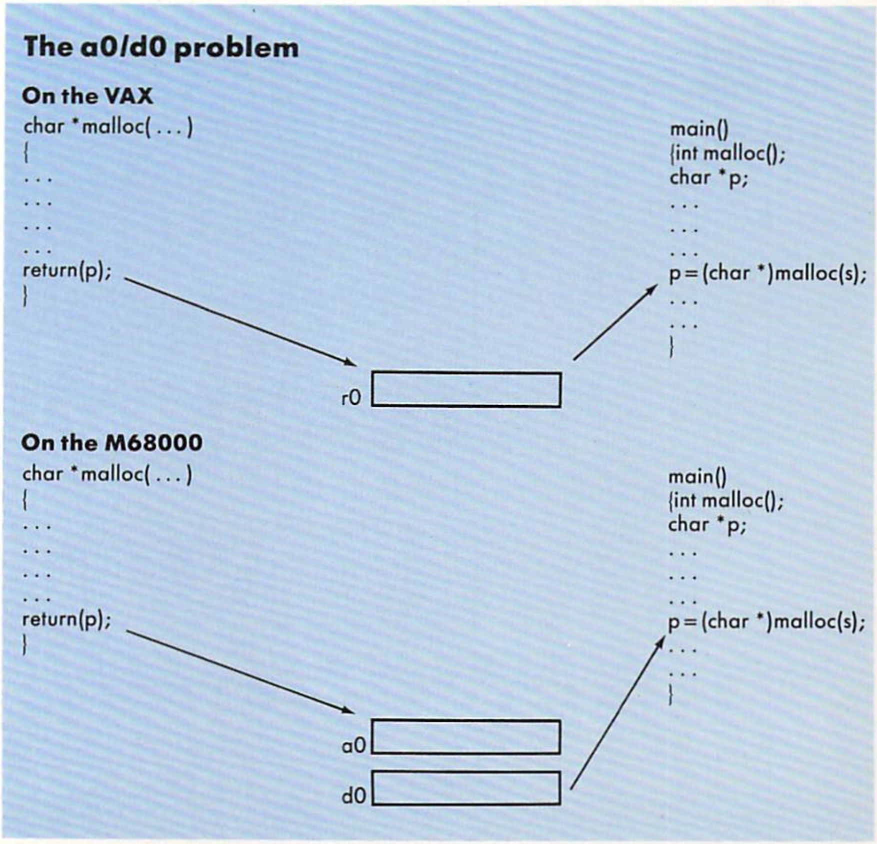
Our grep(1) search caught most occurrences of the hibyte-lobyte problem. It was a nuisance to fix, however. In the case of some utilities such as sdb(1), whose code is already complicated by being reworked by several generations of programmers, it took a lot of study to determine exactly how to fix the byte-order dependencies. The writing of byte-order dependent code is just a bad coding practice. It can and should be avoided by all C programmers.

Another ramification of this problem was that any files containing multibyte entities such as *short*s or *int*s required selective byte-swapping before being downloaded to be run on the M68000 target. This was not too difficult for COFF files, since we had a utility to do exactly this, but some data files used by the utilities were more troublesome. We finally decided to create many of them on the target machine instead of downloading them from the host.

The utilities affected by this hibyte-lobyte problem included prs(1), tplot(1G), fsdb(1M), unpack(1), and sdb(1).

a0/d0 problem. An interesting portability problem was the a0/d0 function return problem. On the VAX, all function return values in C are left in the r0 register by the compiler. Hence it does not matter on the VAX whether a function is declared, say, to return an *int* or a pointer, the returned value will always be in r0. The M68000 compiler, however, returned *int*s in the d0 data register and pointers in the a0 address register. Mismatched function declaration/invocations thus caused the return of garbage for us.

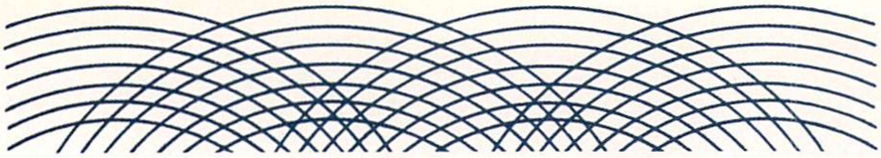
Figure 1 shows how this can happen. The procedure *malloc()* is a frequently used library function that returns a pointer to some free space. Its definition is represented on the left. It is shown as being invoked by the program on the right which has, however, an incorrect declaration for it. Instead of:



```
char *malloc( ... )
{
...
...
...
...
return(p);
}
```

```
main()
{int malloc();
char *p;
...
...
...
p=(char *)malloc(s);
...
...
}
```

Figure 1.



```
char *malloc();
```

it has:

```
int malloc();
```

(Actually, the declaration in the calling program was simply missing, but since the type defaults to *int*, the effect is the same.) In the case of the VAX, the returned value was put in the r0 register by *malloc* and was retrieved from that register by the calling program, so all was well in spite of the bug.

In the case of the M68000, however, *malloc* put the returned value (a pointer) in the address register a0 but the calling program looked for it in d0 where a returned *int* would belong. The calling program therefore used garbage from d0 instead of the pointer in a0. (It was later found that the values remaining in d0 were typically numbers like 0, 1, or 2, which were left over from previous temporary calculations and function calls.)

Because of the explicit pointer cast in the assignment:

```
p = (char *)malloc(sizeof(buf));
```

no compiler warning was given. By using these small garbage numbers instead of pointers to legitimate free space, we were effectively using the text (instruction) space at the beginning of user memory as a free memory pool! The cause of the bug was only discovered after read-only text segments were implemented and we experienced segment violations trying to write to write-protected segments. It is perhaps a testimony to the robustness of that UNIX system that programs actually ran fairly well in this state.

Utilities rendered nonportable by this bug included fsck(1M), dc(1), file(1), and getty(1M). Diff3(1) had a similar problem involving an undefined return value, which prevented it from working correctly on the M68000.

Padding problem. Here was another example of a bug that happened to be symptomless on the VAX but was deadly on the M68000. In the nroff(1) code was a structure element declared as *char zz;* in one place and *char *zz;* in another.

On the VAX this was innocuous since the field was padded to a 4-byte boundary on that machine and the *char* took up as much space as the pointer. The M68000 SGS II C compiler, however, pads to 2-byte boundaries. The *char* type is allocated 2 bytes and the pointer is allocated 4, so that all items in a data table following *zz* are off by 2 bytes. The symptom produced by this bug was rather amusing The characters output from nroff(1) were shifted one position in the ASCII map. For example, F became G, P became Q, and Hello World became Ifmmp Xpsme.

Figure 2 illustrates, in a simplified form, what happened. At the top of Figure 2 are two structure declarations, inconsistent with each other. In the first, *zz* is declared as a *char* type and actually occupies 1 byte of storage. The VAX compiler forces the following array to start at a 4-byte boundary, however, so 3 bytes of padding are inserted between *zz* and the array *a*. When the entire structure is dumped to a file, it therefore occupies 1 + 3 + 26 = 30 bytes.

If the file is then read in by a program that thinks the structure is as defined on the right, 4 bytes are read in for *zz* (now a pointer) and 26 bytes are read in for the array *a*. The original 30 bytes are read back in and everything is hunky-dory. Again, the bug is masked.

The bottom of Figure 2 shows what happens on the M68000. In the structure on the left, *zz* is 1 byte as before, but now, since the M68000 only requires padding to a 2-byte boundary, only 1 byte of padding is added. The total number of bytes written out is 28. When the program using the structure declaration on the right then

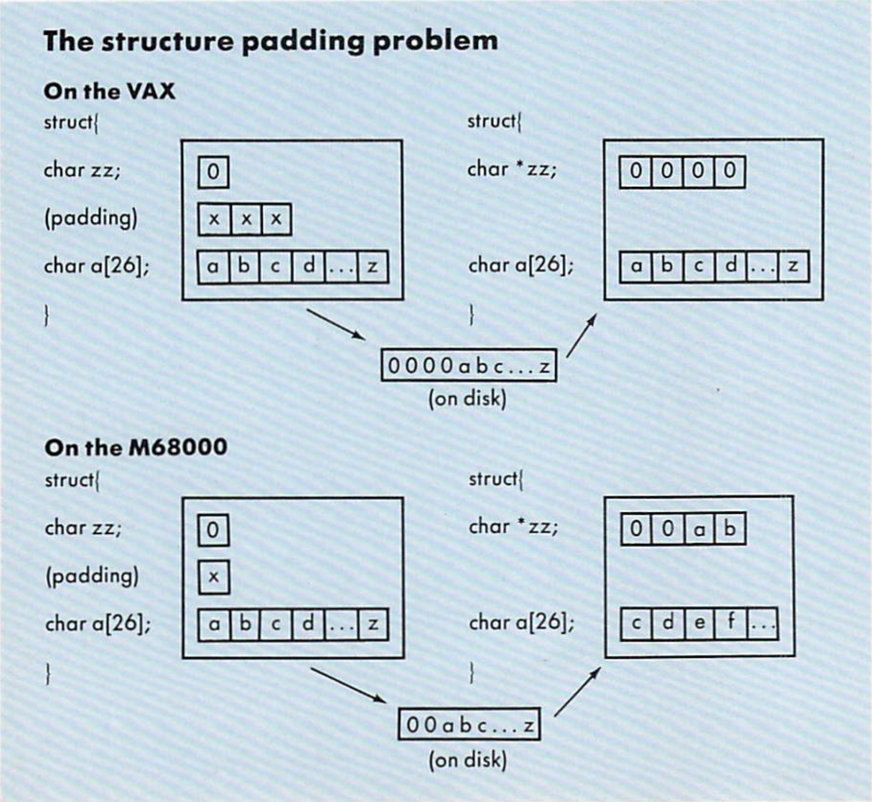
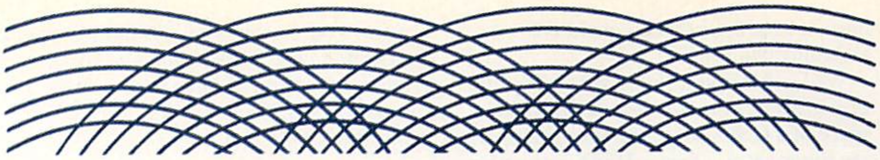


Figure 2.



tries to read the disk, it wants to read 30 bytes. After reading in 4 bytes for the pointer, it now reads in the array *a*, which is consequently shifted by 2 bytes over the original *a*.

The nroff(1)/troff(1) system contains about 180 files and working on this problem consumed at least 10 man-days just before we were scheduled to send SYSTEM V/68 to AT&T for approval. This bug nearly delayed submission of our ported system. What made it difficult to find was that it was a bug present in the VAX code but symptomless because of a coincidence. On the M68000, that coincidence no longer held.

The padding problem also complicated our debugging of the symbolic debugger sdb(1). In initial stages of our work, we were debugging this utility on the VAX by transporting core-dump files created on the M68000-based machine back to the VAX. We had to write a small utility to massage the user structure within the core file to the VAX-style structure padding.

Flexname problem. Syntactically, our SGS II cross compiler for the M68000 differed in several ways from the resident C compiler on the VAX. For example, our new compiler supported arbitrary-length identifiers, with all characters considered significant, whereas the resident VAX compiler considered two identifier names the same if they agreed in the first seven characters.

One might think that the new feature, which in a sense subsumed the old, would not cause any portability problems. However, here is what happened: the old code would contain two variable references such as *baseaddress* and *baseaddr* with the intent that they refer to the same thing. Under the new compiler, they would no longer be considered the same and problems would arise. I have always hated the compiler feature that allows one to use long identifiers but only attaches significance to the first few. This confirmed my dislike.

Utilities affected by this problem were uucp(1C), sdb(1), expr(1), wall(1M), mail(1), and newform(1).

Prepended underscore problem. The resident compiler on the VAX prepended an underscore to all C variable names, and these augmented names were passed on to the assembler and ended up in the symbol table. Our SGS II compiler did not prepend underscores. Utilities that had to be modified because of this difference were ipcs(1), ps(1), fuser(1M), and sdb(1), all of which looked at symbol table information.

sdb(1). The port of this utility is a singular case since it is extremely machine dependent, involving disassembly, setting breakpoints, single stepping and reading object modules and core files. The port of this utility alone took about one man-year (see "Experiences in Porting UNIX SDB to the M68000" in references).

sh(1). Although the shell was surprisingly portable, it had a few problems—most minor but one serious. One minor problem was that our compiler no longer accepted the obsolescent, ambiguous C constructs =+ and =-. These had to be changed to the currently accepted += and -=.

On a more serious note, the shell had an unusual way of expanding its data segment at run time. It simply kept making data accesses until it generated a memory fault signal, then caught the signal, requested more memory, and proceeded. On the M68000, it was not possible for us to catch this signal. We therefore had to check for possible memory faults before exceeding the segment limit.

Null pointer ref problem. In C, a string variable is a pointer. It points to a sequence of bytes terminated by a 0 byte. This is represented in Figure 3(a). A variable representing a null string is thus a pointer to a zero byte, for example:

```
s[0] = '\0';
```

makes the value of *s* a null string. This is depicted in Figure 3(b). The assignment:

```
s = 0;
```

is something quite different. It makes the string pointer *s* point to memory location 0. The value of the string itself is whatever happens to be at memory location 0. If we try to print this string out with the statement:

```
printf("%s",s);
```

we would get a string of ASCII characters representing whatever was at location 0. The string would go on until a null character (0) were encountered.

The UNIX System V code contained something similar to this. The variable *s* was declared as:

```
static char *s;
```

and then, under a certain chain of circumstances, was printed out before anything was assigned to it. (Under C, statics are

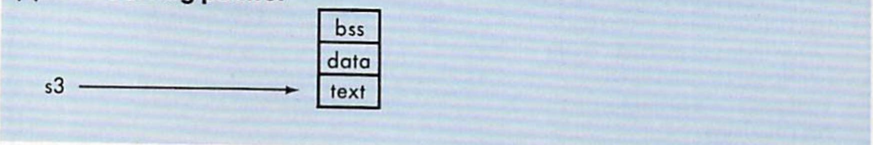

The null pointer reference problem
(a) An ordinary string
s1
T h i s i s a s t r i n g 0
(b) A null string
s2
0
(c) A zero string pointer
bss
data
s3
text

Figure 3.

initialized at 0, so this is equivalent to assigning 0 to *s*.)

Well, what is at location zero? Location zero is always the first word of the text space of a UNIX program—for example, see Figure 3(c). (Text is UNIX jargon for instruction space.) On the VAX, the first byte of text space is a register mask which always happens to be 0. Therefore, it just so happens that an uninitialized static string is null and will not appear as garbage when printed out.

This is, however, bad coding practice and is another example of a hidden bug waiting to cause problems when the code is transported to another machine. If one prints *s* out on the M68000, there happens to be no register mask, and one gets garbage in the form of instructions interpreted as ASCII characters. This affected the utility awk(1).

Few problems due to C

I do not want to give the impression with this paper that I think C has substantial portability problems. The fact that we went through several hundred thousand lines of code and found only a few portability problems attributable to C or C coding practices is amazing.

Most of the C portability problems we found were really in the form of latent bugs—that is, bugs which were masked on the original system by some coincidence and were revealed only after the port. These include the a0/d0, structure padding, flexname, and null pointer reference problems. In some cases, however, such as the hibyte-lobyte issue, the problem was not covert bugs but simply poor coding practices. The hibyte-lobyte problem can and should be avoided.

Of the 250 utilities mentioned at the beginning of this paper, 207 were found to be literally source-code portable from the VAX to the M68000-based EXORmacs development system. Most of these problems were related to machine architecture and minor compiler differences. Experience in finding and dealing with these problems greatly facilitated further porting work we did, such as the UNIX System V Release 2 port.

References

Dion, L.C., and Filipski, A.J. "Project Source Code Control for the UNIX Operating System—Part I." *COMPUTER LANGUAGE* (Nov. 1984).

Dion, L.C., and Filipski, A.J. "Project Source Code Control for the UNIX Operating System—Part II." *COMPUTER LANGUAGE* (Dec. 1984).

Filipski, A.J., and Carey, L. C. "Experiences in Porting UNIX SDB to the M68000 Processor." *Proceedings of the Second Conference on Software Engineering*. Nice, France: AFCET (1984).

Hergenhan, C.B.; Kretsch, D.J.; and Wehr, L.A. "UNIX System V Port Acceptance Criteria." Internal document, Bell Laboratories, Murray Hill, N.J. (1984).

Jalics, Paul J., and Heines, Thomas S. "Transporting a Portable Operating System: UNIX to an IBM Minicomputer." *Communications of the ACM* (Dec. 1983): 1066-1072.

Johnson, S.C., and Ritchie, D.M. "Portability of C Programs and the UNIX System." *Bell Systems Techical Journal* 57, 6, (July-Aug. 1978).

Legg, G. "Portable Operating Systems Create Common Program Environment." *EDN* (Sept. 29, 1983).

Pawlowski, B.J., and Filipski, A.J. "The Dynamics of a Semi-Large Software Project with Specific Reference to a UNIX System Port." *Proceedings of the Summer USENIX Conference* (June 1984).

Tilson, M. "Moving UNIX to New Machines." *BYTE* (Oct. 1983).

Yates, J. L. "UNIX and The Standardization of Small Computer Systems." *BYTE* (Oct. 1983).

UNIX is a trademark of AT&T Bell Labs.

Alan Filipski holds a Ph.D. in computer science from Michigan State Univ. He has taught at Central Michigan Univ. and Arizona State Univ. and is currently a principal staff engineer at Motorola Microsystems, Tempe, Ariz., working on the UNIX System V operating system.

Advanced Screen Management made easy

Now a professional software tool from Creative Solutions.

WINDOWS FOR C™

More than a window display system, WINDOWS FOR C is a video tool kit for all screen management tasks.

- Pop-up menus and help files
- Auto memory management
- Keyboard interpreter
- Word wrap
- Auto scroll
- Highlighting
- Color control
- Overlay and restore
- Plus a library of over 50 building block subroutines

Designed for enhanced portability. Easy to learn, easy to use.

Once you've tried WINDOWS FOR C, you'll wonder how you ever managed without it.

Full support for IBM PC/XT/AT and compatibles, plus interfaces for non-IBM computers; Lattice C, CI-C86, Mark Wm. C, Aztec C, Microsoft C, DeSmet C (PC/MSDOS),

NEW Ver. 3.1
Enhanced portability.
Topview compatible.

WINDOWS FOR C $195
(specify compiler & version)
Demo disk and manual $ 30
(applies toward purchase)

Full source available.
No royalties.

Creative Solutions
21 Elm Ave., Box T4,
Richford, VT 05476
802-848-7738
Master Card & Visa Accepted
Shipping $2.50
VT residents add 4% tax.

CIRCLE 14 ON READER SERVICE CARD

COMPILERS

AMSTERDAM COMPILER KIT

Another in a series of productivity notes on UNIX™ software from UniPress.

Subject: A complete Kit of compilers, cross compilers and assemblers.

The Amsterdam Compiler Kit is the only C and Pascal UNIX package which includes a wide range of native and cross tools. The Kit is also easily modifiable to support custom targets.

Features:

- *C and Pascal compilers (native and cross) for UNIX machines.*
- *Host and target machines include VAX™ 4.1/4.2 BSD, PDP™-11/V7, MC68000™ and 8086.™ Cross assemblers provided for 8080,™ Z80,™ Z8000,™ 8086,™ 6800,™ 6809,™ 68000,™ 6502 and PDP-11.*
- *The Kit contains complete sources* of all programs, plus comprehensive internals documentation on how to make modifications needed to add a new program language or new target machine.*

**A source UNIX or C license is required from AT&T.*

Price:

Full Source System	*$9950*
Educational Institutions	*995*

Selected binaries are available – contact us with your machine type.

For more information on these and other UNIX software products, call or write: UniPress Software, Inc., 2025 Lincoln Hwy., Edison, NJ 08817. Telephone: (201) 985-8000. Order Desk: (800) 222-0550 (Outside NJ). Telex: 709418. Japanese Distributor: SofTec 0480 (85) 6565. European Distributor: Modulator SA (031) 59 22 22

OEM terms available.
Mastercard/Visa accepted.

UNIX is a trademark of AT&T Bell Laboratories. VAX & PDP 11 are trademarks of Digital Equipment Corp. MC68000, 6800 & 6809 are trademarks of Motorola Corp. 8080 & 8086 are trademarks of Intel Corp. Z80 & Z8000 are trademarks of Zilog, Inc.

CROSS COMPILER
FOR THE 8086™ FAMILY

LATTICE® C CROSS COMPILER

Another in a series of productivity notes on UNIX™ software from UniPress.

Subject: C Cross Compiler for the 8086 Family.

The Lattice C Cross Compiler allows the user to write code on a VAX™ (UNIX or VMS™) or MC68000™ machine for the 8086 family. Lattice C is a timesaving tool that allows a more powerful computer to produce object code for the IBM-PC™. The compiler is regarded as the finest C compiler for the 8086 family and produces the fastest and tightest code.

Features:

- *For your UNIX or VMS Computer.*
- *Use your VAX or other UNIX machine to create standard Intel object code for the 8086 (IBM-PC).*
- *Highly regarded compiler produces fastest and tightest code for the 8086 family.*
- *Full C language and standard library, compatible with UNIX.*
- *Small, medium, compact and large address models available.*
- *Includes compiler, linker, librarian and disassembler.*
- *8087™ floating point support.*
- *MS-DOS™ 2.0 libraries.*
- *Send and Receive communication package optionally available. Price $500.*
- *Optional SSI Intel Style Tools. Package includes linker, locator and assembler and creates executables for debugging on the Intel workstation or for standalone environments. Price $8,550.*

Price:

VAX (UNIX or VMS)	*$5000*
MC68000	*3000*

For more information on these and other UNIX software products, call or write: UniPress Software, Inc., 2025 Lincoln Hwy., Edison, NJ 08817. Telephone: (201) 985-8000. Order Desk: (800) 222-0550 (Outside NJ). Telex: 709418. Japanese Distributor: Softec 0480 (85) 6565. European Distributor: Modulator SA (031) 59 22 22.

OEM terms available.
Mastercard/Visa accepted.

Trademarks of: Lattice, Lattice, Inc.; VAX and VMS, Digital Equipment Corp.; UNIX, AT&T Bell Laboratories; IBM PC, International Business Machines; MS DOS, Microsoft; MC68000, Motorola; 8086/8087, Intel

UniPress Software
Your Leading Source for UNIX Software.

CIRCLE 81 ON READER SERVICE CARD

SuperSoft Languages
When Performance Counts

A programmer's most important software tool is the language compiler or interpreter he uses. He has to depend on it to work and work well.

At SuperSoft, we believe it. That's why we offer four excellent compilers: SuperSoft FORTRAN, SuperSoft A, SuperSoft C, and SuperSoft BASIC. They answer the programmer's need for rock solid, dependable performance on microcomputers.

SuperSoft FORTRAN

With large code and data.
SuperSoft FORTRAN version 2.0 with large code and data space is now available under MS DOS and PC DOS. It gives you the power to compile extremely large FORTRAN programs on micros. It allows double precision and complex numbers, full IEEE floating point, and a full range of other important features for the serious FORTRAN programmer. Both 8087 support and a RATFOR preprocessor are optionally available.

FORTRAN (CP/M-80 & 86, MS DOS, PC DOS): $325
8087 support: $50 RATFOR: $100

SuperSoft A

A true Ada* subset
SuperSoft A is a completely standard subset of the Ada language, incorporating approximately 63% of the standard Ada syntax and including such important features as packages and separate compilation. For CP/M-80 microcomputers: $300.

SuperSoft C

SuperSoft C is a high-powered, full-featured C compiler designed for serious C applications. It is fast – both in compilation and execution, and it is packed with more than 200 library functions (all delivered in source code form). SuperSoft C produces optimized assembly code, and object code can be ROMed.

SuperSoft C (for CP/M-80, CP/M-86, MS DOS, PC DOS): $350

Languages
BASIC Compiler
Fortran
C Compiler
SuperSoft

Languages
BASIC Compiler
SuperSoft

SuperSoft BASIC

The SuperSoft BASIC compiler lets you get serious with business and financial programs. It uses BCD math to give you highly accurate results for demanding applications. SuperSoft BASIC is a true native code compiler that is generally compatible with Microsoft's BASIC interpreter. And an additional bonus – no run time license fee is required.

SuperSoft BASIC Compiler (for MS DOS, PC DOS, and CP/M-86): $300

Also available for programmers:

Star-Edit, a full-featured programmer's text editor: $225.00
Disk-Edit, an invaluable programmer's disk data editor: $100.00

To order call: **800-762-6629**

In Illinois call **217-359-2112**

SuperSoft

SuperSoft, Inc., 1713 S. Neil St.,
P.O. Box 1628, Champaign, IL 61820

In conjunction with SuperSoft, SuperSoft FORTRAN was developed by Small Systems Services, Urbana, IL, a leader in FORTRAN development.

Japanese Distributor: ASR Corporation International, TBL Building, 7th Floor, 1-19-9 Toranomon, Minato-Ku, Tokyo 105, Japan Tel. 03-5025550. Telex 222-5650 ASRTYO J.

*Ada is a trademark of the Department of Defense
PC DOS is a trademark of International Business Machines.
MS DOS is a trademark of Microsoft.
CP/M-80 and CP/M-86 are trademarks of Digital Research, Inc.

CIRCLE 74 ON READER SERVICE CARD

A spreadsheet becomes a development environment

Symphony Command Language

By Darryl Rubin

How is a spreadsheet program like a piano? To many musicians, the piano is the ultimate instrument—an orchestra for a single player. Many microcomputer users perceive the spreadsheet the same way: it is their all-purpose computing tool.

Multifunctioned spreadsheets like Lotus 1-2-3 have been used to model everything from business expenses to Conway's Game of Life and to perform everything from number crunching and business graphics to simple word processing. Using 1-2-3's macro facility, some people have even built templates that can almost—but not quite—serve as turnkey application packages.

Now Lotus Development Corp., the maker of 1-2-3, has given us Symphony, a combination spreadsheet, data base, word processor, graphics, and communications package. Most importantly, this prodigious program offers what 1-2-3 lacked for building turnkey applications: a true programming language called, appropriately, Symphony Command Language (SCL).

In this article we'll audition SCL and learn how to write SCL macros. We'll describe one of the program's bugs plus a significant but undocumented feature. And for an encore, we'll present some macros that are surprisingly useful and entertaining. (You'll find these selections and more in the file SCL.WRK on the *COMPUTER LANGUAGE* Bulletin Board Service and CompuServe CLM SIG.)

So quiet, please, the performance is beginning.

Classical quintet

Lotus calls Symphony an integrated program and, indeed, Symphony includes what has quickly become the classical set of five functions for this category of software: spreadsheet, word processing, data base, graphics, and telecommunications.

What makes Symphony unique is how it ties these five functions together. Other integrated programs implement their major functions as distinct environments between which you must explicitly transfer data. Not Symphony. This performer implements all its functions within a single environment—a gigantic spreadsheet—so when you switch from one function to another what you're really doing is changing your mode of operating on the same, underlying data.

And you can change modes as easily as a musician changes keys. Symphony offers multiple windows that you can independently pan, zoom, and switch from mode to mode. You may use multiple windows to view the same data in different ways simultaneously. What's more, you can write SCL macros that manipulate the underlying spreadsheet data without concern for how the data is being viewed.

SCL programs can even manipulate themselves, because in true von Neumann style, they are stored in the spreadsheet along with the data they operate on. Think of Symphony as a virtual computer whose memory is a rectangular array of storage cells— the spreadsheet!

Like the cells of any memory, those of the spreadsheet have both a value and an address.

A cell's value can be one of two types: numeric or string ("label," in spreadsheet parlance). Furthermore, a value can be either constant (a literal) or calculated (a formula). For example, 3 and "Alpha" are literals, while +1+2 and "Al"&"pha" are formulas.

Cell addresses can also be of two types: coordinate or range. A coordinate is simply the row/column location of a single cell, while a range is the pair of coordinates at opposite corners of a block of cells. For example, A1 and Z100 are coordinates, while A1 . . Z100 is a range of 2,600 cells.

The nicest thing about cell addresses in Symphony is that you can name them symbolically. You could, for instance, define AVAR as a name for cell A1 and ANARRAY as a name for the range A1 . . Z100. Doing this is analogous to declaring variables in languages like BASIC and Pascal. In fact, the spreadsheet implements not just the program and data store for SCL but also its symbol table.

The Symphonic formula

The soul of any spreadsheet is, of course, the formula. Formulas are what make a spreadsheet come alive and be responsive to the data you enter into it. But formulas in Symphony have an additional, higher purpose: they constitute the expression syntax of SCL.

SCL's expression syntax is the standard, algebraic one used in almost every language from FORTRAN to Ada. Included are the full set of arithmetic and comparison operators, plus string concatenation (&) and Booleans (*#AND#, #OR#, #NOT#*). Here are some example expressions:

```
(2^3)+1        ;Result is 9
A1>A2          ;False if A1=2, A2=3
A1*NDX         ;6 if NDX is A2's name
C1&"!"         ;"Oh!", if C1 = "Oh"
```

SCL also sports a handsome repertoire of built-in functions, about 90 in all. Included are a full set of transcendental, logical, string, statistical, financial, and date/time functions. Let's highlight a few of the more unusual (for a spreadsheet) of these.

Symphony's string functions are one of its strengths. You can use these to nicely format spreadsheet titles, build models that operate on string data, and write SCL macros that work in the word processing mode.

Lotus seems to have borrowed the string functions primarily from BASIC. You get *@LEFT, @RIGHT,* and *@MID* for extracting substrings, *@FIND* for searching a string (like BASIC's *INSTR*), *@REPT* for repeating a string, *@LENGTH* for determining a string's length, and *@VALUE* for converting strings to numbers.

The logicals seem to have been inspired by C. *@ISSTRING* and *@ISNUMBER* let you test whether an argument is a string or a number, while *@ISERR* will tell you whether or not the argument expression produces a valid value. In a similar vein, *@ISNA* lets you check whether or not a value is available (i.e., whether a cell is blank).

You can test these and other conditions using the function *@IF*:

```
@IF(COND,THEN_EXPR,ELSE_EXPR)
```

This function returns the value of the *THEN_EXPR* if the *COND* expression is "true" (nonzero); otherwise it returns the value of the *ELSE_EXPR*. For example, the following expression returns a cyclically incremented value based on an input value in A1 and a limit value in a cell named LIMIT:

```
@IF(A1+1>LIMIT,1,A1+1)
```

The *@IF* function is actually a specialized form of the more general *@CHOOSE(I,EXPR1, . . EXPRN)*, which returns the value of the *I*th expression in the argument list. You can use this to implement a simple but crude kind of array access, as in:

```
@CHOOSE(I,A1,A2,A3,A4,A5)
```

Another way to do the same thing is to use the *@INDEX* function, like so: *@INDEX(A1 . . A5,0,I)*. This function implements two-dimensional array lookup, returning the value of the cell at a specified row/column offset from the origin of the specified range. The range can be explicit, as in the preceding example, or symbolic: *@INDEX(ANARRAY, COL,ROW)*. The macros we present later put this function to excellent use.

@INDEX is one of several functions that accept range specifications. Others include *@ROWS* and *@COLS*, which tell you the dimensions of a specified range, and the statistical functions *@SUM, @MIN, @MAX,* and *@AVG*. For example, *@SUM(ANARRAY)* will sum all the cells in the named range. Or try this out:

```
AVG(ANARRAY,B101 .. B105,ZZZ)
```

Yes, the statistical functions even accept a variable number of arguments. And I'm sure you'll have no argument with that!

Macro maestro

Now that we've mastered formulas, the question is, how do we make them play? Formulas are fine for writing expressions in SCL, but how are control structures expressed? Can we go beyond mere spreadsheet recalculation to create an orchestrated sequence of programmed actions?

Indeed we can. Enter the spreadsheet macro. Lotus 1-2-3 introduced this concept. In that program, as in Symphony, a macro is basically a set of keystrokes recorded as labels in the spreadsheet.

Suppose you have a cell named ACM that contains the string *Association for Computing Machinery*. In word processing mode, you could hit the user macro key (F7) followed by ACM<CR> and watch the acronym type itself. You've made a WP glossary feature out of a macro.

As you can see, a Symphony macro is simply a label stored in a named range. This range can be a single cell, as in the preceding example, or a vertical column of them, as in Listing 1.

When you invoke a macro, Symphony interprets the first cell in the named range, then drops vertically down and interprets the next cell, etc.; the crossing of cell boundaries is immaterial. Execution stops when Symphony encounters a blank cell or the macro directive {*RETURN*}.

You polished 1-2-3 users may gasp at a perfectly readable directive like {*RETURN*}. Gone are 1-2-3's cryptic */X* macro commands. Replacing them are a much enhanced set of statements whose general syntax is:

```
{KEYWORD ARG1,ARG2, .. ARGN}
```

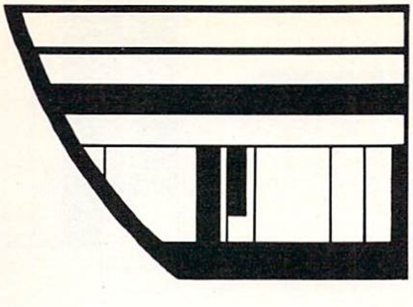
Symphony provides two classes of macro keywords: key names and control statements.

Key names are symbolic names for nonalphanumeric keys like the cursor controls and function keys. These include {*UP*}, {*DOWN*}, {*LEFT*}, {*RIGHT*}, {*PGUP*}, {*PGDN*}, {*ESC*}, and {*BS*}. All key names take an optional, numeric argument that serves as a repeat count. For example, the following label in a macro cell will type the string *BRAVO!* and then backspace over it:

```
BRAVO!{BS 6}
```

Some of the other key names you can use are {*ZOOM*}, {*CALC*}, {*MENU*}, and {*SERVICES*}. The last two are especially important because they invoke Symphony's two top-line menu bars, {*MENU*} for the mode-specific menu bar and {*SERVICES*} for the global services bar. The macros we'll discuss later give these key names a real workout, but here's a preview: {*MENU*}*IR* ~.

Obviously Lotus has left some parts of SCL cryptic! But if you know Symphony, you'll recognize this as the keystroke sequence to perform the *Insert Row* function (the tilde is interpreted as a carriage return). The inverse function to delete a row is {*MENU*}*DR* ~.



Unfortunately, you'll have to watch out if you want your macros to insert and delete rows or columns like this, because Lotus's programmers missed a beat when writing this version of Symphony. You see, Symphony doesn't check whether spreadsheet insertion or deletion shifts an executing macro horizontally or vertically; if it does, execution resumes in, ahem, the wrong cell.

Besides key names, SCL provides a set of control statements for branching, conditional execution, and looping. Branching is as simple as writing {*BRANCH LOC*}; control transfers immediately to the specified cell. Conditional execution looks like this: {*IF COND*}. If *COND* is true, execution continues in the same cell, otherwise it skips to the cell below. Shades of BASIC's limited *IF*! This certainly isn't the block-structured *IF/THEN/ELSE* you might have expected.

For more of the unexpected, look at this:

```
{FOR CNTR,START,STOP,STEP,LOC}
```

Here we have SCL's loop statement. *CNTR* designates a cell where the loop counter will be stored, *START* is the starting value for this counter, *STOP* and *STEP* should be obvious, and *LOC* is the name or coordinate of a macro that should be used as the loop body. Seem strange? Perhaps, but it makes sense: think of the *FOR* statement as an iterated procedure call.

Speaking of procedures—writing them is a snap, because every macro you define becomes an SCL statement name like any other. For example, Listing 1 shows a pair of macros named {*PUSH*} and {*POP*} that implement a numeric stack within the spreadsheet. Here are some examples of using them:

```
{PUSH 1}         ;Push a literal
{PUSH AVAR}      ;Push a cell
{POP}{POP}       ;Pop both
```

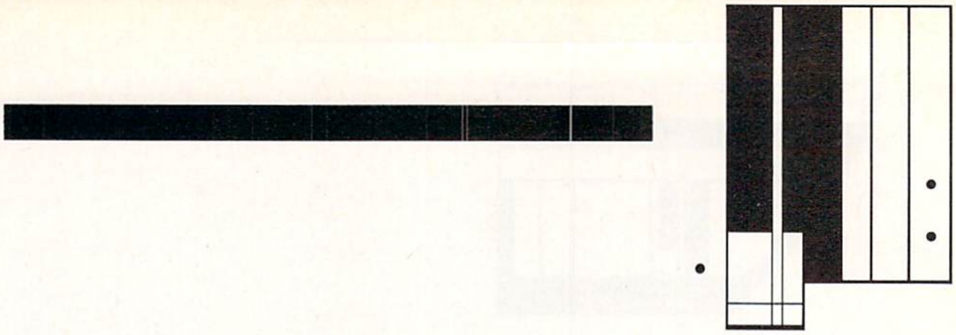
As you can see, {*POP*} takes no argument; it always returns its value in the cell named ELEMENT. This is because argument passing in SCL is by value only; procedures can't pass results back out through the parameters.

These two macros illustrate some more of SCL's statements. First is {*DEFINE LOC*}, which {*PUSH*} uses to specify a cell where the incoming parameter is to be received. Macros that take several parameters can specify multiple parameter cells by saying {*DEFINE LOC1, . . LOCN*}. All incoming parameters are assumed to be strings unless the suffix *:VALUE* is appended to the location name (Listing 1).

The next statement of interest is SCL's assignment statement, {*LET LOC,EXPR*}.

```
{PUSH} and {POP} macros

;NOTE: For this and Listings 2 and 3, labels are in Col A,
;statements in Col B, and comments in Col C. Highlight Col A and
;use Range Name Labels Right command.

element          ; Stack element being pushed or popped
top              ; Top of stack index (preset it to 0)
stack            ; *** MAKE A RANGE "stack" in col B spanning rows as needed:
                 ; (2nd row of stack range)
                 ; *** End of "stack" range (insert rows as needed)

         ; Push -- Push a number on a stack, ie {push num}
push     {define element:value}              ; Define cell for input arg
         {put stack,0,top,element}           ; Put arg on stack, then increment
         {let top,@if(top+1>@rows(stack),@err,top+1)}~        ; top pointer
         {return}

         ; Pop -- Pop a number and return in cell "element", ie {pop}
pop      {let top,@if(top<=0,@err,top-1)}    ; Point back to top element
         {let element,@index(stack,0,top)}   ; Pull it off stack
         {put stack,0,top," "}~              ; Then blank that stack cell
         {return}
```

Listing 1.



It stores the value of the *EXPR* at the specified location. Due to a peculiarity (bug?) of Symphony's, it's a good idea to force a spreadsheet recalculation after every {*LET*} (the tilde after each {*LET*} in Listing 1 accomplishes this if automatic recalculation is enabled). The recalculation is necessary to assure that all references to the assigned cell will immediately reflect the new value.

Listing 1 also illustrates a variant of {*LET*} called {*PUT*}. Its syntax is {*PUT LOC,COL,ROW,EXPR*} and what it does is perform an assignment to an array cell, where *LOC* is the origin (upper left corner) of the array and *ROW* and *COL* are the relative offsets from the origin. This is the inverse of the *@INDEX* function we described earlier, which retrieves a value from an array.

Pictures at an Exhibition

{*PUSH*} and {*POP*} are useful but, ho hum, dull. For something with a bit more flash, check out Listing 2. Here we have a macro that is quite literally flashy, because it implements a slide show!

To try it, create and lay out your slides on the monitor as windows for which you have set Window Settings Restrict Range Screen. Next, set up a named table that lists the slides to be presented (see the SLIDEMO table in Listing 2). Then hit Alt-S, sit back, and enjoy.

You might ask how the macro became bound to the Alt-S key. The answer is the macro's special name, ∖*S*. Change the S to any character from A to Z and the macro would be bound to that Alt key instead.

The first thing the slide macro does is use {*GETLABEL*} to ask where the slide table (i.e., program) is. Much like BASIC's *INPUT* statement, this one issues a prompt and returns a string typed by the user. Its syntax is {*GETLABEL PROMPT,LOC*}, where *PROMPT* is the prompt string and *LOC* is the location where the user's response should be returned.

For inputting numbers rather than strings, SCL has a complementary {*GETNUMBER*} statement. One caution on using either statement, however: you must force a spreadsheet recalculation to ensure that formulas referencing the input cell (*LOC*) immediately reflect the new value (in Listing 2, the tilde after the {*GETLABEL*} statement accomplishes this).

Having asked for the range name of the slide table, the slide macro next uses a {*FOR*} loop to process each entry in that table. Notice how it determines the number of slides to iterate over by using *@COUNT*. This handy function counts the number of nonblank cells in a specified range, which in the case of the slide table will be twice the number of slides to be shown (remember, the slide table has two columns).

@COUNT's general syntax is *@COUNT(RANGE)*. So what's that strange *@@(SLIDES)* argument we're passing to *@COUNT* in Listing 2? It's not a range name nor coordinate pair, obviously; it is an undocumented function, one that provides an important, indirect range naming capability.

Anywhere SCL allows a range name you can code *@@(LOC)* instead, and SCL will use the label contained in *LOC* as the range name. The slide macro uses this function to operate on a slide table specified interactively by the user, as opposed to one hard-coded into the macro. A caution, however: force a recalculation before using *@@(LOC)* to ensure that it accesses the most recent value stored at *LOC*.

The {*FOR*} loop we just analyzed repeatedly calls the macro procedure *SHOWSLIDE*, which does the actual work of showing each slide. *SHOWSLIDE*'s trick is how it uses a {*LET*} statement to copy the current slide name into the body of the procedure. There, the name becomes part of the {*SERVICES*}*WU* (Window Use) command executed by *SHOWSLIDE* to display the slide's window. This is one example of how SCL macros can modify themselves to good effect, but there is an even more powerful way, as we'll see next.

Space opera

Do your spreadsheets lack depth? Do you feel constrained by a table's two dimensions? Then you'll like our finale, a macro that projects Symphony's spreadsheet into three dimensions (Listing 3).

A 3D spreadsheet is a vertical stack of 2D spreadsheets that have the same format but different data—say, expense reports for each month of the year. To create one, first enter a model into the spreadsheet that will serve as a template for creating the stack of data planes. Then hit Alt-T and respond to the prompts for the range name of the model and the depth of the 3D stack to create.

The 3D macro will copy the model plane to create the requested number of data planes in a vertical column below the model. It will then position the cursor to the origin of the first data plane. Each data plane will have a range name of MODELi, where MODEL is the name you specified for the model plane, and "i" is a number from 1 to the number of data planes..

You may now flip from plane to plane (Alt-N, Alt-P) with the cursor staying at the same relative coordinates, or you may put into the current cell a formula that consolidates values from other planes (Alt-C).

The 3D macro makes extensive use of SCL's capability for self-modifying macros. The slide macro presented earlier showed how to modify a macro cell by using {*LET*} to assign it a value. 3D's flourish is to include formulas within the cells of the macro.

We said earlier that a macro is a vertical column of cells containing labels that Symphony interprets as keystrokes and macro commands. But a macro cell can instead contain a label-producing formula, in which case that formula's current value is interpreted by SCL. This lets you write macros whose behavior depends, via embedded formulas, on data elsewhere in the spreadsheet.

The *FILLPLANE* procedure in Listing 3, for example, uses formulas that depend on the *MODEL* and *PLANEI* variables to synthesize a keystroke sequence that will copy the contents of the model plane to the *I*th data plane and then position the cursor at the origin of that data plane.

Listing 3 demonstrates another of SCL's strengths—its user-definable menus. When the statement {*MENUCALL*

Illustration: Barbara Luck

FUNC} is executed by the \ *C* macro, a menu of statistical functions is displayed according to the table named *FUNC*. Menu tables like this have one column for each command in the menu, with the command name in the top cell and its help text in the next cell. The remainder of each column is the macro procedure to be called when that command is picked from the menu.

The macro procedures of the *FUNC* menu are simple, for they are merely part of a keystroke sequence initiated by the \ *C* macro to fill the current cell. The result, however, is that the menu-selected function name becomes part of a formula that makes the cell display the consolidated value of corresponding cells in higher numbered planes. This macro starts simple, but what a finish!

Unfinished Symphony

As full-featured as SCL may appear, it is an unfinished language—intentionally so,

```
Alt-S -- Macro to show a set of slides stored in windows

          ; SAMPLE SLIDE RANGE: first col (col B) has slide (window) name,
          ;                     second (col C) has slide delay (seconds)
slidemo   slide1         3 ; Resize and modify contents of this range
          slide2         3 ; as needed.  Note:  pressing any key will
          slide3         3 ; cancel slide delay and advance to next slide.
          main           1 ; Trailing blank entries in slide range are OK

slides            ; Name of range defining slides to be shown
n                 ; Number of slide currently being shown
kb                ; Cell to store user keystroke
delay             ; Delay loop counter

          ; Main loop:  ask for range containing slide names
          ;             and cycle through them
\s        {getlabel "Enter range containing slide names: ",slides}
          {windowsoff}{paneloff}
          {for n,0,@count(@@(slides))/2-1,1,showslide}; loop thru slides
          {windowson}{panelon}{return}

          ; Routine to show current slide
showslide {let nextslide,@index(@@(slides),0,n)}      ;get slide name
          {services}wu              ; go to slide's window
nextslide                ; <= window (slide) name put in col B by {let} above
          ~{windowson}{windowsoff}  ; put slide on display
          {for delay,0,@index(@@(slides),1,n)-1,1,waitloop}    ; Pause
          {if kb<>""}{get kb}       ; Discard any user keystrokes
          {return}

          ; Routine to delay for specified time or until key hit
waitloop  {blank kb}{look kb}       ; Check for user keystroke
          {if kb<>""}{forbreak}     ; End wait loop if keystroke
          {wait @now+.000001}       ; Else wait a second
          {return}
```

Listing 2.

for Lotus has included in Symphony a unique add-in application interface.

This interface defines a set of entry points and data structures by which independent software vendors, and perhaps ultimately users, can extend Symphony's capabilities or modify existing ones. Add-ins are already available for accessing DOS functions, executing SCL macros from disk libraries, and more.

Even ignoring these possibilities, the part of SCL we covered here is but a prelude to a work of much greater magnitude. SCL has scores of additional features to delight and challenge the practiced programmer.

Indeed, because of its complexity and some of its quirks, SCL is not the easiest language to learn or use. Like the piano, it can be a versatile and expressive instrument . . . it just takes a little practice. ■

Darryl Rubin is section manager for network products at ROLM Corp.

```
Alt-T -- Make a 3D spreadsheet from a model (template) range

model             ; Name of model range for creating planes
depth             ; Total number of planes to create
planevars ; CREATE A RANGE "planevars" in col B spanning the rows
          ; containing nexti, previ, planeI, planerow, and planecol.
          ; These cells contain formulas as follows:
i                 ; Index (z coord) of current plane (set by macros below)
nexti             ; @if(i+1>depth,1,i+1)   <= Calcs index of next plane
previ             ; @if(i-1<=0,depth,i-1)  <= Calcs index of prev plane
planeI            ; +model&@string(i,0)  <= Calcs name of current plane
          ; Next 2 formula cells maintain plane-relative cursor coords
planerow          ; @cellpointer("row")-@cell("row",@@(planeI))
planecol          ; @cellpointer("col")-@cell("col",@@(planeI))
coordstr          ; String form of plane coords, set by \c macro

          ; MAIN ROUTINE:  Ask for model range name and 3-D sheet depth
\t        {getlabel "Enter model range: ",model}~
restart   {if @iserr(@rows(@@(model)))}{branch notfnd}; Branch if bad range
          {getnumber "Enter 3-D sheet depth: ",depth}
          {windowsoff}{paneloff}                ; suppress screen refresh
          {goto}                                ; go to model range
                  ; <= Formula in Col B: +model
          ~{for i,1,depth,1,makeplane}          ; loop to make all planes
          {let i,1}~                            ; point to first plane
          {goto}                                ; go to first plane
                  ; <= Formula in Col B: +planeI&"~"
          {windowson}{panelon}                  ; enable screen refresh
          {return}
          ; Reprompt user until valid model range name is given
notfnd    {getlabel "Hit CR, highlight range, then hit CR again ",depth}
          {menu}rnc                             ; Create range for user
                  ; <= Formula in Col B: +model
          ~{?}~                                 ; Let user highlight range
          {branch restart}                      ; Back to mainline

          ; Routine to create and fill new plane from model
makeplane {down @rows(@@(model))}   ; advance to origin of next plane
          {newplane}                ; create range for the plane
          {fillplane}               ; copy model to newly created plane
          {return}

          ; Routine to create range for next plane
newplane  {menu}rnc                             ; do Range Name Create
                  ; <= Formula: +planeI         ;  using name of new plane
          ~~{menu}rnd                           ; Now delete the range and
                  ; <= Formula: +planeI         ; recreate it.  This hackery
```

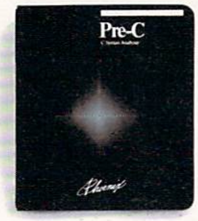
Listing 3 *(Continued on following page).*

```
          ~{menu}rnc                         ; is needed in case range
                  ; <= Formula: +planeI&"~." ; already existed (don't ask).
          {down @rows(@@(model))-1}          ; Anchor top left of range
          {right @cols(@@(model))-1}~        ; Anchor bottom right

          ; Routine to copy model plane to new plane
fillplane {menu}c ; Invoke Copy command
                  ; <= Formula: +model&"~"  (source of copy)
                  ; <= Formula: +planeI     (dest of copy)
          ~{goto} ; New plane is now filled, go to its origin
                  ; <= Formula: +planeI     (range name to go to)
          ~{return}

          ; Alt-N:  Go to next plane, preserving relative cursor coords
\n        ~{windowsoff}{paneloff}{recalc planevars}
          {goto}  ; Go to origin of next plane
                  ; <= Formula in Col B: +model&@string(nexti,0)
          ~{down planerow} ; Move cursor to same relative coords
          {right planecol} ;  in new plane as in current plane
          {let i,nexti}~   ; Make next plane the current plane
          {windowson}{panelon}{return}

          ; Alt-P:  Go to prev plane, preserving relative cursor coords
\p        ~{windowsoff}{paneloff}{recalc planevars}
          {goto}  ; Go to origin of previous plane
                  ; <= Formula in Col B: +model&@string(previ,0)
          ~{down planerow} ; Move cursor to same relative coords
          {right planecol} ;  in new plane as in current plane
          {let i,previ}~   ; Make prev plane the current plane
          {windowson}{panelon}{return}

          ; Alt-C:  Put consolidation formula in current cell
\c        {recalc planevars}        ; Ensure state reflects current coords
          {if i>=depth}{beep}{return}        ; Quit if in bottom plane
          {let argument,"("}        ; Initialize formula argument string
          {let coordstr,@string(planecol,0)&","&@string(planerow,0)}
          {windowsoff}{menu}e~      ; Erase current cell in plane
          {push i}                  ; Save current plane index
          {edit}@{menucall func}             ; Enter function into plane
          {for i,i+1,depth,1,bldargs}        ; Append argument string
argument                            ; <= Arg string built in col B by BLDARGS
          {bs})~{windowson}                  ; Terminate formula string
          {pop}{let i,element}~{return}      ; Restore plane index & exit

          ; Routine to append arguments to consolidation formula
          ; Each arg has the form @index(planename,planecol,planerow)
bldargs   {recalc planevars}
          {let argument,+argument&"@index("&planeI&","&coordstr&"),"}
          {return}

          ; Menu to prompt for consolidation function.  Note: define a
          ; range "func" spanning cols B through H in the next two rows.
func      Sum     Average  Minimum  Largest  Count    DeviationVariance
          Sum the Average tMinimum oMaximum oCount theStandard Variance of
          sum     avg      min      max      count    std      var
          {return}{return} {return} {return} {return} {return} {return}
```

Listing 3 *(Continued from preceding page).*

Programmers' Pfantasies™ by Phoenix.

Phoenix makes programmers' dreams come true. With the best-engineered, highest performance programming tools you can find. A full line of MS-DOS® /PC DOS programs and utilities no other company offers. All designed to help you write, test and deliver the best programs possible. Top-of-the-line quality at a price you can afford.



Finally, A Lint For MS-DOS.

Now you can get the full range of features C programmers working in UNIX™ have come to expect from their Lint program analyzer.

With Pre-C™ you can detect structural errors in C programs five times faster than you can with a debugger. Find usage errors almost impossible to detect with a compiler. Cross-check multiple source files and parameters passed to functions. Uncover interface bugs that are difficult to isolate. All in a single pass. Capabilities no C compiler, with or without program analyzing utilities, can offer. In fact, Pre-C outlints Lint, since you can handle analyses incrementally.

Pre-C's flexible library approach lets you maintain continuity across all the programs in your shop, whether you use Pre-C's pre-built libraries, pre-existing functions you already have, or some you might want to buy yourself.

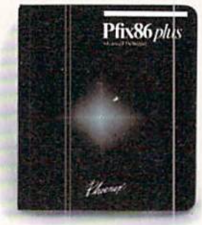
Plus, you're not limited to one particular library, and Pre-C keeps track of all the libraries you're using to make sure that code calls them correctly. **$395.**

Pasm86
Phoenix

Assemble Programs Twice As Fast.

Pasm™86 will assemble MASM files two to three times faster than MASM 3.0. Pasm86 supports 8086/88, 8087, 80186 and 80286 processors.

With Pasm86's built-in defaults, you can write code quickly since you won't spend hours learning all the control statements needed at the beginning of your program. You can define symbols on the command line. Decide whether you want error messages or not. And, put local symbols within procedures. **$295.**



Still Fixing Bugs The Hard Way?

Pfix™86 Plus, the most advanced symbolic debugger on the market, eliminates the endless error searches through piles of listings. Locate instructions and data by symbolic name, using symbolic addresses. Handle larger, overlayed programs with ease.

An adjustable multiple-window display shows source, object code and data, breakpoint settings, current machine register and stack contents simultaneously. An in-line assembler allows program corrections directly in Assembly language. Powerful breakpoint features run a program full speed until a loop has been performed n times.

With a single keystroke you can trace an instruction and the action will be immediately reflected in source, object, data, stack, and register windows. Another key begins a special trace mode that executes call and loop instructions at full speed. Designed to work with both Plink™86 and MS© LINK linkage editors. **$395.**

Ptel
Phoenix

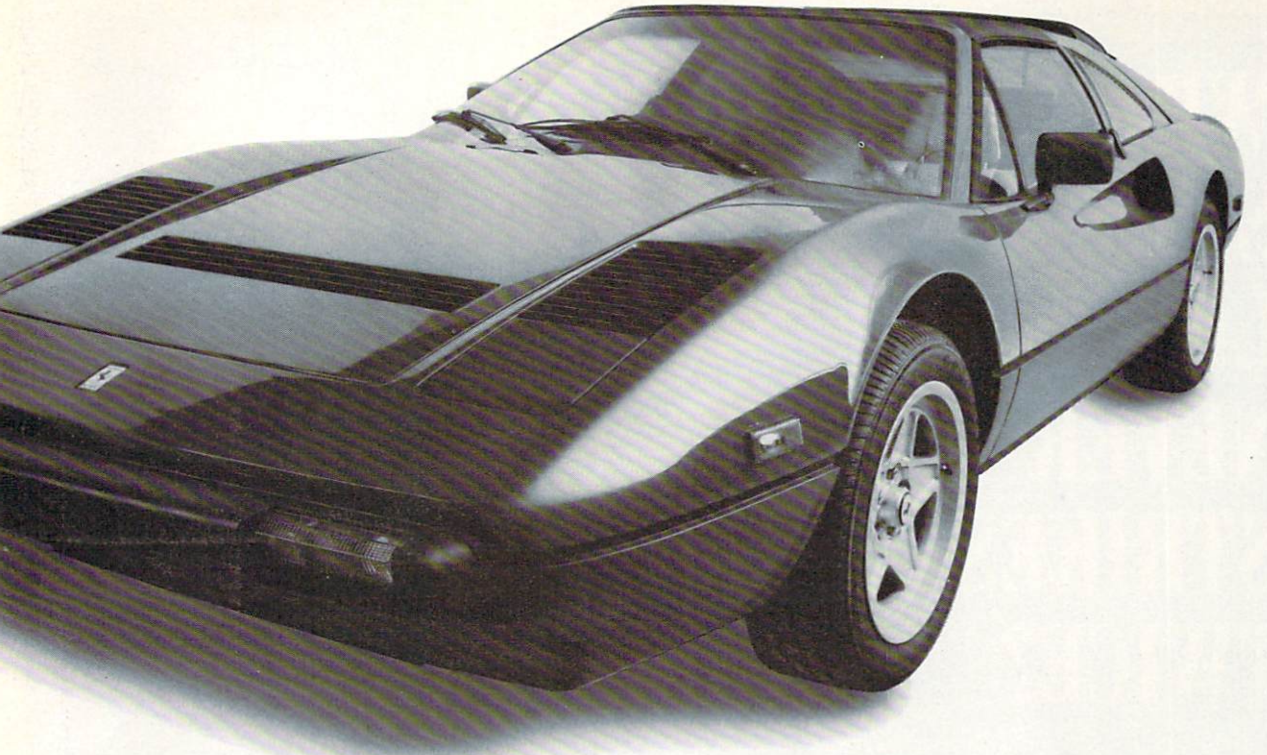
Get The Lead Out Of Binary File Transfer.

Ptel™ is the universal binary file transfer program for MS-DOS 2.0 or higher. You can move binary files fast and accurately. Upload or download groups of files from Bulletin Boards or remote computers. Move files between dissimilar machines and operating systems. Ptel's advanced binary protocol, Telink,™ offers better-than-Modem7 accuracy and performance. Faster transfer speeds. An on-screen update of error correction, blocks transferred, and transmission time.

Includes popular Modem7 and XModem protocols. With checksum or CRC. Plus Kermit and ASCII. **$195.**

Special Thanks to Gasten Andrey of Framingham, MA.

Programmer's Pfantasies, Pre-C and Pfinish are trademarks of Phoenix Computer Products Corporation.
MS-DOS and MS are registered trademarks of Microsoft Corporation.
IBM is a registered trademark of International Business Machines Corporation.



Maximize Your Program's Efficiency.

Pfinish™ delivers the fastest running programs possible. This performance analyzer lets you "zoom in" on the inefficient parts of your program. Whether written in Assembly language. C. PASCAL. FORTRAN. BASIC. Unlike profilers available today, Pfinish understands the structure of your program and reports the amount of activity and time spent in its subroutines or functional groups. Pfinish analyzes both overlaid and memory resident programs. Down to the instruction level. Reports are displayed. Stored on disk. Or printed out. In tabular form or histograms.

Do a dynamic program scan. Identify the most frequently executed subroutines. Find inefficient code that costs your program valuable time. Rank subroutines by execution frequency. **$395.**

Why Work With A Primitive Editor?

More than a powerful editor, Pmate™ is a text processing language. An emulator of other editors. A language-specific editor for C, PASCAL, and FORTRAN. Pmate can even run in the background!

You get full-screen, single-key editing. Ten editing buffers. Horizontal and vertical scrolling. A "garbage stack" buffer. A built-in macro language with variables, control statements, radix conversion, tracing and 120 commands that you can group and execute with a single keystroke. **$225.**

XT and AT are trademarks of International Business Machines Corporation.
UNIX is a trademark of AT&T Bell Laboratories.
Pasm86, Pfix86 Plus, Ptel, Telink, Pmate and Plink86 are trademarks of Phoenix Software Associates Ltd.
Intel is a registered trademark of Intel Corporation.

Why Squeeze Your Program More Than You Have To?

The Plink86 overlay linkage editor brings modular programming to 8086/88-based micros. Write large and complex programs without worrying about memory constraints. Work on modules individually, link them into executable files. Use the same module in different programs. Change the overlay structure of an existing program without recompiling. Use one overlay to access code and data in other overlays. **$395.**

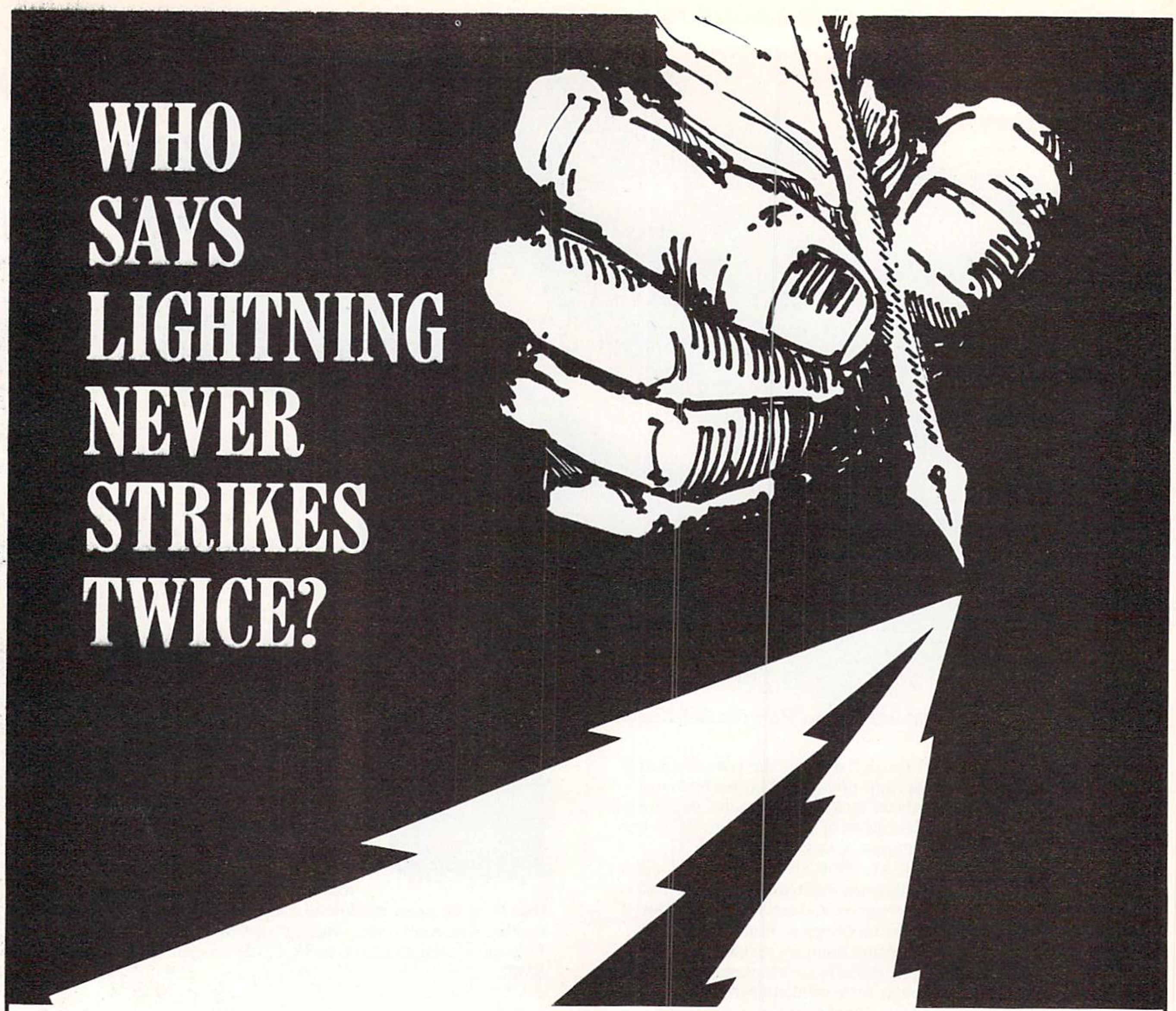
Call (800) 344-7200. In Massachusetts (617) 762-5030. Or, write.

Phoenix

Phoenix Computer Products Corp.
1420 Providence Highway Suite 115
Norwood, MA 02062

Outside the US, contact: Lifeboat Japan, Tokyo, Japan, Telex #2423296 LBJTYO, Telephone: 03-456-4101 • Repro Haganum (Stevis BV), Dinther, Netherlands, Telex #39581 REHAGNL, Telephone: 01-720-74543, 01-720-75543 • Memory Data, Sundbyberg, Sweden, Telephone: 46-8764-6700 • Roundhill Computer Systems, Marlborough, Wiltshire, UK, Telex #444453 AWARE G, Telephone: 44-467254675.

CIRCLE 65 ON READER SERVICE CARD


WHO SAYS LIGHTNING NEVER STRIKES TWICE?

Whitesmiths sets yet another precedent
with the *first* in a series of new portable standard C Compilers:

C for the 8086 family

Features:

- Includes Pascal which conforms to *full* ISO (level 1) standard, plus popular extensions and long identifiers
- C now has struct assignment, enumerations, plus other popular features and long identifiers
- Supports all memory models from small to large, *plus mixed pointer sizes and segment overrides*
- Source level portable debugger included
- Generates assembler listings with intermixed source code
- Multi-segment linker with direct *or* sequential libraries, plus librarian, assembler, and other object tools included
- Source code of system interface library included
- Library use fees included in purchase price

Whitesmiths, Ltd. ◆ 97 Lowell Road ◆ Concord, MA 01742 ◆ 617-369-8499 ◆ Telex 750246

Getting to Know PL/I

By Gary Sarff

PL/I (Programming Language I) has been in existence for nearly 20 years now and so may be considered one of the old-timers' languages. Some programmers who use Ada today may not know what they owe to PL/I, but language designers recognize some of the philosophy and feel of PL/I in Ada. This article is intended to present the flavor of this powerful and sometimes maligned language.

Until the late 1970s PL/I was a language that was not commonly seen outside of universities and data processing departments of large organizations. Because it was such a large language, it was too difficult to implement on small computers and remained a language for mainframes—until PL/I subset G was introduced by ANSI and made implementations for microcomputers feasible. Subset G sacrifices some of the facilities and features of full PL/I, but the programmer can usually create any needed features as procedures, so the loss is not keenly felt.

Until the advent of Ada, PL/I was probably the epitome of language design by committee. Nearly every imaginable feature was thrown into the language and the compiler kept getting larger and larger. Compared to the lean elegance of Pascal, PL/I is considered by some purists of language design to be a monstrosity. It has been called unteachable, unlearnable, and inelegant. Some of these complaints, as with any language, are about a particular implementation of PL/I, but many are justified.

PL/I is implemented as a multipass compiler—the compiler will scan through the programmer's source code from beginning to end many times, generating all sorts of intermediate code and tables. This is quite different from Pascal, for example, which is usually a single-pass compiler. PL/I is simply too large and has too few restrictions on some things, such as declarations, to allow it to be a single-pass compiler. The number of passes varies with the implementation and the compiler options specified, but it is rumored that on IBM mainframes PL/I compiles in anywhere from 96 to 150 passes!

Nevertheless, PL/I is a quite useful and powerful language for the business as well as the scientific programmer. It was designed to be the language to replace all languages. It contains features of FORTRAN, COBOL, and ALGOL, three of the most commonly available languages of the time.

PL/I has separate procedures with local variables, a block structure, recursion, record variables, a full range of arithmetic and string operations, and numerous file I/O options. Recursion is not quite as transparent to the programmer as in more modern languages such as Pascal and Ada. A procedure must be declared to be recursive when it is written, or a recursive call will be considered an error. This is really a concession to the optimizer so that it can produce better code for certain types of procedure calls.

Statements

A statement in PL/I has the form:

```
{LABEL:} {KEYWORD} {STATEMENT
     OPTIONS};
```

where all parts are optional. Labels are used to designate the destinations of *GOTO* statements, and the first character should be a letter. According to various IBM manuals and text books, PL/I has in excess of 380 keywords, statements, and options. Statements are terminated by semicolons, and more than one statement may appear on a single line. A complete program will contain a main procedure and probably some declarations.

Declarations consist of declarations for variables and declarations for subroutines and subprograms. PL/I distinguishes subroutines as procedures internal to a program written in the same language as that program, while subprograms are external, probably compiled separately from the main program and written in a different language.

Unlike Pascal, PL/I declarations may be placed anywhere in the source program mainly because PL/I is a multipass compiler. The first declaration is of the program that is to be compiled. It is of the form:

```
program_name: PROCEDURE
     OPTIONS(MAIN);
```

where *OPTIONS(MAIN)* signifies that this is the main program. Other entries valid in the *OPTIONS* option are FORTRAN, COBOL, and Assembler, which are used to link together subprograms written in those languages to a PL/I program.

The last statement must be:

```
END program_name;
```

to signify the end of the source program. Between these two statements, the user's subroutines, variable declarations, and main program can appear in virtually any order.

Data types

In the area of variable declarations, PL/I offers a rich assortment of data formats to the user. As a legacy from FORTRAN, undeclared variables have a default type based upon the first letter of the variable name. Variables starting with *I* through *N* are whole numbers (integer type). Variables starting with *A* through *H* and *O* through *Z* and the symbols *@*, *#*, *$* are considered to be floating point numbers. The explicit types of declarations for variables are made with the PL/I *DECLARE* statement (Figure 1).

For numeric variables, the base attribute may be either *DECIMAL* or *BINARY*. The scale attribute may be *FIXED* or *FLOAT* (denoting fixed point or integer and floating point or real). The precision attribute may have different allowable ranges depending upon the implementation. The declaration in Figure 1 specifies that variable *PRICE* is of the form XXXX.XX (six digits, two of which must be after the decimal point).

Declarations can cause confusion to beginning learners of PL/I because the language is so flexible. Numerous defaults are set by the compiler if the programmer does not specify explicitly what he or she wants. Also, the order of key words in statements is not strictly laid out.

```
DECLARE  PRICE DECIMAL FIXED (6,2);
   |         |         |      -----
   |         |         |        |______Precision attribute
   |         |         |_____________Scale attribute
   |         |_______________________Base attribute
   |_________________________________Variable name
```

Figure 1.

Figure 2 presents more examples of declarations.

The *BINARY* attribute causes numbers to be stored in their binary representation. This usually increases the speed of computations since simpler hardware is needed to perform arithmetic on binary numbers. The indices for loops and subscripts to arrays are usually declared to be fixed binary for speed and efficiency.

Also from FORTRAN comes the declare mode attribute *COMPLEX* to declare a complex number. A sample declaration is:

```
DECLARE NUM1 FIXED DECIMAL
      (8,3) COMPLEX;
```

A variable called *NUM1* will now exist with the *COMPLEX* attribute where both the real and imaginary parts of *NUM1* have precision XXXXX.XXX. The *COMPLEX* attribute can be used freely with the other base and scale attributes to form *COMPLEX FLOAT BINARY (14)* and so on. Such variables have values assigned in the natural notation of mathematics such as *NUM1* = 4 + 3*I*, which will put 4 in the real part and 3 in the imaginary part. Arithmetic can be performed just as with all real variables, and a variety of functions such as *CONJG(X)* returns the complex conjugate of the complex number *X*.

In deference to COBOL and the business programmer, PL/I offers the *PICTURE* declaration as follows:

```
DECLARE GRAND_TOTAL PICTURE
      '99,999.99';
```

where the period specifies decimal point alignment and the nines specify that a digit (0-9) may appear here.

The declaration is the same as:

```
FIXED DECIMAL (7,2)
```

in terms of precision and arithmetic results. The internal storage is one character or byte for each digit of the number, one byte for the comma, and one byte for the decimal point. Picture declarations are used mostly for formatted I/O, similar to the *PRINT USING* statement found in many versions of BASIC. The difference is that arithmetic may be performed on a picture variable.

Arrays are declared in a straightforward manner.

```
DECLARE TEMP(1:10,1:10) FIXED
      BINARY(31);
```

makes *TEMP* a two-dimensional array with subscripts running from 1 to 10.

Character strings are declared in two different ways:

```
DECLARE STR CHAR(20);
```

makes *STR* a character string with a fixed length of 20. Shorter strings placed in *STR* will be left justified and padded with blanks. PL/I also has a dynamic length string, declared as follows:

```
DECLARE TWOSTR CHAR(20) VARY-
      ING;
```

which means the maximum length of *TWOSTR* is 20, but it will change length to accommodate the data placed into it.

PL/I also provides pointer variables as in Pascal, but they are not differentiated. A variable is declared *POINTER*, meaning it will hold a machine address, but it may hold the address of any PL/I data structure, not just a single type as in Pascal.

As a really exotic feature, some implementations of PL/I provide data types of various nations' currencies. A program may have numeric literals with an implied type of *STERLING* (U.K. currency).

Assume the variable *AMOUNT* is declared as follows:

```
DECLARE AMOUNT FIXED DECIMAL
      (7,0);
```

then the following assignment to *AMOUNT* of a *STERLING* literal:

```
AMOUNT = 3.4.2L;
/* Interpret as 3 pound 4 shilling 2
     pence */
```

will result in *AMOUNT* having the value of 770 interpreted as British pence.

PL/I also allows programmer control over the storage allocation attribute of a declaration. By default, variables are of storage type *AUTOMATIC*, which means that they behave as variables behave in a block structured language— created and destroyed as execution enters and leaves various blocks. Variables may also be declared *STATIC*, in which case they always exist independently of the block they may be declared in. Or they may be declared *CONTROLLED*, in which various "generations" of a variable are maintained by PL/I automatically on a push-down stack. The program only has access to the topmost incarnation of the variable, but new values can be pushed (using the statement *ALLOCATE(variable_name)*) and popped (using the statement *FREE(variable_name)*).

PL/I is one of the few languages to provide a programmer accessible stacks as a data structure. A complete sample program is shown in Listing 1. *GET LIST* is used for input and *PUT LIST* for output of values. Comments start with /* and proceed until a */ is encountered.

Notice the assignment of variables and calculations to variables of differing types and precisions. PL/I provides such conversions automatically and indeed is quite happy to oblige the programmer by attempting to convert any type to nearly any other type, *FIXED* to *FLOAT*, *DECIMAL* to *BINARY* to *PICTURE*, *CHARACTER* to *FIXED* to *BIT* strings, and many more. This is an anathema to proponents of strongly typed languages like Pascal and can admittedly cause many hard-to-track-down bugs, especially when the compiler does not inform the programmer what it is doing.

Loops

Loops in PL/I can be a confusing subject because so many obscure loop control statements can be written. PL/I has the normal iterative loop called a *DO* loop of the form:

```
DO I= 1 TO 10; or
DO I= X TO 1 BY −1; or
DO J = −3 TO 100 BY 4;
```

Floating point values can also be used:

```
DO STEP = 0.0 TO 5.0 BY 0.1;
```

PL/I also has an enumerated *DO*, as in:

```
DO COUNT = 1 TO 10 BY 2,17 TO 33
     BY 3,50 TO 100 BY 5; or
DO I= 4,5,11,-2,7,19,1;
```

In the first example, *COUNT* will first go from 1 to 10 by 2, then from 17 to 33 by 3, and last from 50 to 100 by 5. In the second example, *I* will successively take on the values 4, 5, 11, −2, 7, 19 and 1. So the loop will be executed seven times.

Conditional *DO* loops are also provided:

```
DO WHILE(J<SIN(ANGLE)); or
DO UNTIL(X>50.2);
```

The difference is that with *DO WHILE*, the test is made at the top of the loop, and with *DO UNTIL*, the test is made at the bottom, so *DO UNTIL* always executes the body of its loop at least once. The various kinds of *DO* loops can be used in one control statement (connected by an implied logical *AND*), and here lies a source of confusion, not only for the programmer but sometimes for the compiler as well. For example, a valid complicated *DO* loop could be:

```
DO I=4,5,7,−2,ABS(X+Y),40 TO
     50 BY 3 WHILE(J>I*5+2)
     UNTIL(I<X*Y−30);
```

Now quick, what is the terminating condition of this loop? Such things may be useful in certain cases, but they probably cause more problems debugging than they could possibly be worth. It is also possible to place *DO* loop specifications into I/O statements, similar to FORTRAN, such as:

```
GET LIST((VALUES(I) DO I=1 TO
     NUMBER_OF_ITEMS));
```

```
DECLARE   PI  FLOAT DECIMAL(6);   {Floating point with 6 significant digits}
DECLARE   BIG FLOAT BINARY(31);   {BIG will be a 31 bit number stored
                                   internally as a binary number}
DECLARE   RATE FIXED BIN(7,2);    {BIN is a PL/I abbreviation for binary
                                   RATE is stored as XXXXX.XX where the
                                   point is a binary point not a decimal
                                   point}
```

Figure 2.

A program to print out Fibonacci numbers in PL/I

```
PROG1: PROCEDURE OPTIONS(MAIN);
       /* THIS PROGRAM WILL READ A USERS'S INPUT AND PRINT OUT THE
          FIBONACCI NUMBERS FROM 1 TO UP_LIMIT.
       */
       DECLARE  UP_LIMIT            FIXED DECIMAL (4,0),
                I                   FIXED BINARY(15),
                FOUT                FIXED BINARY(31);

           /* FOUT IS ADDED JUST TO ILLUSTRATE AUTOMATIC CONVERSION
              OF VALUES IN PL/I.
           */

       DECLARE  FIBON               ENTRY RETURNS(FIXED DEC(9,0));

       /* THE ABOVE STATEMENT INFORMS THE COMPILER THAT FIBON IS THE
          NAME OF AN ENTRY POINT TO A FUNCTION THAT RETURNS A VALUE OF
          TYPE FIXED DECIMAL(9,0)
       */

       PUT LIST ("ENTER A NUMBER, TYPE 0 TO STOP");
       GET LIST (UP_LIMIT);
       DO WHILE(UP_LIMIT > 0);
          DO I=1 TO UP_LIMIT;    /* CONVERSION OF UP_LIMIT TO BINARY */
             FOUT=FIBON(I);     /* CONVERSION OF RESULT TO FIXED BINARY */
             PUT LIST(FOUT);    /* CONVERSION OF BINARY TO ASCII CHAR'S */
          END;
          PUT LIST ("ENTER A NUMBER, TYPE 0 TO STOP");
          GET LIST (UP_LIMIT);
       END;
FIBON: PROCEDURE(NUM) RECURSIVE; RETURNS(FIXED DECIMAL (9,0));
       DECLARE   NUM               FIXED BINARY(15);

       IF NUM=1 | NUM=0 THEN            /* THE | IS LOGICAL OR */
          RETURN(1);
         ELSE
          RETURN(FIBON(NUM-1)+FIBON(NUM-2));
       END FIBON;
END PROG1;
```

Listing 1.

which is called an implied *DO* and is the same in execution as:

```
DO I = 1 TO NUMBER_OF_ITEMS;
    GET LIST (VALUES(I));
END;
```

which will read in *NUMBER_OF_ITEMS* numbers from input into the array *VALUES*.

Subroutines and functions

PL/I provides subroutines and functions like many other languages. A subroutine is declared as follows:

```
routine_name:PROCEDURE; or
routine_name:PROCEDURE(arg_list);
```

If the subroutine is recursive, the key word *RECURSIVE* must appear before the semicolon. Subroutines and functions are terminated by an end statement with a name matching the name of the subroutine:

```
END routine_name;
```

If the subroutine takes arguments, they must be declared somewhere inside the body of the subroutine. In PL/I, arguments are always passed as call by reference, (*VAR* in Pascal). If the argument in a *CALL* statement is a literal value, the compiler will create an internal dummy argument and pass the address of that to the subroutine.

One of the facilities of PL/I, the lack of which is often bemoaned in Pascal, is the ability to pass arrays and character strings without regard to their size. Suppose we want to pass a two-dimensional array to the subroutine *SUB1*. We would say in our calling routine:

```
CALL SUB1(TOTALS);
```

assuming *TOTALS* is the name of an array. In the declaration of *SUB1* we would write, for example, the following code:

```
SUB1:PROCEDURE(TEMP);
    DECLARE TEMP (*,*) FIXED
        DECIMAL (6,2);
        ...
END SUB1;
```

The asterisks in the declaration of *TEMP* signify dimensions to be filled in at each run-time execution of this subroutine.

PL/I provides the built-in function *DIM(X,n)*, which returns the number of elements in the *n*th dimension of the array *X*, *HBOUND(X,n)* and *LBOUND(X,n)*, which return the high and low bounds of the *n*th dimension of array *X*.

Functions are distinguished from subroutines by the key word *RETURNS* in their heading definition. The key word *RETURNS* specifies what the type of the returned value will be. Values to be returned are sent through the *RETURN* statement inside the function body. If the value computed in the *RETURN* statement is not of the type specified in the *RETURNS* attribute, the value will be converted to the appropriate type if possible. This is another potential for bugs.

An unusual feature of PL/I that sets it off from other mainframe languages is the ability for the programmer to catch and process interrupts. Most mainframe systems so isolate the user from the computer that such a facility in PL/I is noteworthy. This feature in PL/I is referred to as a condition signal. It is a sort of asynchronous, nonlocal subroutine call. It is asynchronous because the programmer is not usually able to predict when the interrupt will occur and nonlocal because the statements executed in response to the interrupt need not be accessible from the currently executing block according to the scoping rules of PL/I.

These two characteristics can cause no end of trouble for the debugger. The programmer informs the compiler that he or she wishes to process error conditions in the program by the use of an *ON* statement. One of the most useful is the condition to check for end-of-file on input:

```
ON ENDFILE(SYSIN) <statements>;
```

where SYSIN is the name of the standard input file. Any file name may appear in its place and, in fact, multiple *ON ENDFILE*

NEW FEATURES

(Free update for our early customers!)

- Edit & Load multiple memory resident files.
- Complete 8087 assembler mnemonics.
- High level 8087 support.
  Full range transcendentals
  (tan, sin, cos, arctan, logs and exponentials)
  Data type conversion and I/O formatting.
- High level interrupt support.
  Execute Forth words from within machine code primitives.
- 80186 Assembler extensions for Tandy 2000, etc.
- Video/Graphics interface for Data General Desktop Model 10

HS/FORTH

- Fully Optimized & Tested for:
  IBM-PC IBM-XT IBM-JR
  COMPAQ EAGLE-PC-2
  TANDY 2000 CORONA
  LEADING EDGE
  (Identical version runs on almost all MSDOS compatibles!)
- Graphics & Text
  (including windowed scrolling)
- Music - foreground and background
  includes multi-tasking example
- Includes Forth-79 and Forth-83
- File and/or Screen interfaces
- Segment Management Support
- Full megabyte - programs or data
- Complete Assembler
  (interactive, easy to use & learn)
- Compare

BYTE Sieve Benchmark		jan 83
HS/FORTH 47 sec	BASIC	2000 sec
w/AUTO-OPT 9 sec	Assembler	5 sec
other Forths (mostly 64k)		70-140 sec

FASTEST FORTH SYSTEM AVAILABLE.

TWICE AS FAST AS OTHER FULL MEGABYTE FORTHS!

(TEN TIMES FASTER WHEN USING AUTO-OPT!)

HS/FORTH, complete system only: $250.

VISA Visa Mastercard

Add $10. shipping and handling

HARVARD SOFTWORKS

PO BOX 69
SPRINGBORO, OH 45066
(513) 748-0390

CIRCLE 47 ON READER SERVICE CARD

Z80 System Owners Join The 16/32 Bit Revolution Through Evolution

Starting At
$695.00

CO-PROCESSING

The most cost effective way for Z80 system owners to obtain 16/32 bit processing power and software compatibility is via the HSC CO-16 Attached Resource Processor.

CO-16 is compatible with any Z80 system running CPM 2.2 or CPM 3. A few examples include:
- KAYPRO 2/4/10 • TRS 2/3/12/16
- AMPRO LITTLE BOARD
- HEATH 89 • SUPERBRAIN
- XEROX 820 • TELEVIDEO 802/803
- MORROW • EPSON QX-10
- LOBO • OSBORNE 1/EXEC
- CROMEMCO • Plus many more

CO-16

Every CO-16 is delivered with • 16/32 bit micro processor • 16 bit Operating System • 256 Kilo RAM • Z80 interface • 16 bit RAM disk driver • CPM80 2.2 RAM disk driver • CPM 2.2 or CPM 3 compatibility • sources with tools • hardware diagrams • board level or case with power supply. .

CO-1686

The only Z80 16 bit co-processor includes • INTEL 8086 • 6Mhz no wait states • MSDOS 2.11 • IBM BIOS emulator • Memory expansion to 768K • 8087 math co-processor • 3-channel Real Time Clock • Runs many IBM PC applications • Shares hard disk space with CPM80 • PC diskette compatilibility on many systems • CPM86 • Concurrent CPM is coming.

CO-1668

The only Z80 16/32 bit co-processor includes • MOTOROLA 68000 microprocessor • 6 Mhz no wait states • CPM68K • Full "C" compiler with UNIX V7 library and floats • Memory expansion to 1.25 million bytes • NS16081 math co-processor • Real Time Clock • Complete software development environment • 100% file compatible with CPM80 • OS9/68 UNIX look alike coming in February.

Dealer, Distributor and OEM's invited

Hallock Systems Company, Inc.
267 North Main Street
Herkimer, N.Y. 13350
(315) 866-7125

HSC

CIRCLE 12 ON READER SERVICE CARD

statements may appear, one for each file being processed in the program.

After execution of this statement, whenever an end-of-file condition is raised on the specified file anywhere in the currently executing program, control will pass to the statements associated with *ON ENDFILE* and execution will continue there. When the statements have all been executed, control will return to the point where it was interrupted and continue there if possible. The usual practice is to have *ON ENDFILE* set some variable flag:

```
ON ENDFILE(SYSIN)
    END_OF_FILE=0;
END_OF_FILE=1;
GET LIST(X);
DO WHILE(END_OF_FILE);
    ... <do something with X>
    GET LIST(X);
END;
```

Thus the programmer is relieved of checking for end-of-file explicitly.

PL/I also has many *ON* conditions, such as *ENDPAGE* for end of a printed page, *FIXEDOVERFLOW, ZERODIVIDE, SUBSCRIPTRANGE*, and on and on. The programmer is even able to create his or her own conditions:

```
ON MYCONDITION <statements>;
```

and later in the program the programmer can write something like:

```
IF X>Y THEN SIGNAL
    MYCONDITION;
```

which will then act just as if an interrupt for *MYCONDITION* had been raised.

PL/I precompiler

Programmers familiar with C and its preprocessor appreciate the power and usefulness of such a facility. PL/I has a precompiler which is more powerful than C's. It implements many of the major features of the language itself.

It is possible to write entire programs in the precompiler. Precompiler statements are preceded by a percent sign (%). With *%DECLARE* we can declare precompiler variables and then perform various computations on them using arithmetic operators. For precompiler flow control PL/I has *%DO* and *%GO TO*, *%IF %THEN %ELSE*, and even precompiler functions:

```
routine_name:%PROCEDURE;
```

that can be invoked by a compile time function reference. In fact, the precompiler has to do almost as much work as the actual compiler does.

Everything to everybody

PL/I was chosen by IBM to be *the* programming language for its System 360/370 computers, intended to remove the need for programmers to use FORTRAN, COBOL, ALGOL, and Assembler. As we know, these languages are still in use. It seems possible that the design of Pascal with its relative paucity of functions was in part influenced by the perception that PL/I was too complicated and a new language should be more elegant, verifiable, and simpler to implement.

Now we are seeing the backlash against Pascal with the advent of Ada, which is certainly a very powerful language. But Ada is seen by some as another PL/I, a monstrous language by committee with thousands of features that can be misused, conversions of data types, and overloading of operators.

As a sideline for anyone who's interested, on the *COMPUTER LANGUAGE* Bulletin Board Service and on this magazine's account on CompuServe I've uploaded a file called PLI.CAL, which is a good example of many of the PL/I coding techniques discussed in this article. The program will print a calendar for any month after the year 1800 up to the year 9999.

PL/I is a powerful, useful language that still has many proponents. In a sense it can be said to be the most popular language in the world since IBM's operating systems are written in a subset of PL/I. People still write large PL/I programs because the language provides the power many programmers desire. ■

References

A PL/I Primer. IBM Document C28-6808-0 (1967).

Hughes, Joan K. *PL/I Structured Programming*. Wiley & Sons Inc., (1979).

Hoare, C.A.R. "The Emperor's Old Clothes," *BYTE*, (Sept. 1981): 414-425.

Gary Sarff graduated from Eastern Illinois Univ. with a B.S. in computational mathematics. He is currently working toward a master's degree in computer science at Southern Illinois Univ.

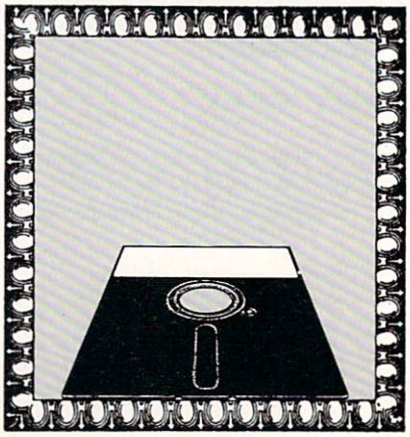


Illustration: Anne Doering

mbp COBOL
for your IBM/PC

The new standard for convenience.

ANSI '74
COBOL
Compiler
mbp
Common
Business
Oriented
Language

Now, the mbp COBOL Compiler offers unrivaled convenience to go with its unmatched performance.

Here are the convenience features you've wished for:

1) an enhanced Screen Management System with program-controlled video attributes and color; **2)** support for PATH & sub-directories; **3)** DOS command execution from within a COBOL program; **4)** 'permanent' DEFAULT modification.

The new mbp Compiler has them all! And they're exclusives: you get them *only* with mbp.

Plus, it's 4 times faster.

Because the mbp COBOL Compiler generates native machine language object code, it executes programs *at least* 4 times faster (see chart). Now, we've made that performance even more convenient to use.

GIBSON MIX Benchmark Results
Calculated S-Profile
(Representative COBOL statement mix)
Execution time ratio

mbp* COBOL	Level II* COBOL	R-M* COBOL	Microsoft* COBOL
1.00	4.08	5.98	6.18

*128K system with hard disk required. IBM PC & AT are IBM TMs; Netware is a Novell, Inc. TM; Level II is a Micro Focus TM; R-M is a Ryan-McFarland TM; Microsoft is a Microsoft TM.

The complete COBOL.

An Interactive Symbolic Debug Package included standard; Multi-keyed ISAM structure; SORT & CHAIN; GSA certification to ANSI '74 Level II; IBM/PC-AT and TI Professional compatibility; with mbp, you get it all. Optional: Novell NetWare interface.

mbp COBOL: the choice of professionals.

It's no surprise more and more companies like Bechtel, Bank of America, Chase, Citicorp, Connecticut Mutual, Hughes Aircraft, McDonnell-Douglass, and Price-Waterhouse choose mbp COBOL.

Make it your choice, too. Just send the coupon, or call, for complete information. Today.

mbp COBOL.$1000

Please send complete mbp COBOL information to:

NAME ____________________

COMPANY ____________________

ADDRESS ____________________

CITY/STATE/ZIP ____________________

PHONE ____________________

mbp Software & Systems Technology, Inc.
7700 Edgewater Drive, Suite 360
Oakland, CA 94621
Phone 415/632-1555

mbp®

CIRCLE 53 ON READER SERVICE CARD

THE PROGRAMMER'S SHOP™

helps compare, evaluate, find products. Straight answers for serious programmers.

SERVICES

- **Programmer's Referral List**
- **Compare Products**
- **Help find a Publisher**
- **Evaluation Literature free**
- **BULLETIN BOARD - 7 PM to 7 AM 617-826-4086**
- **Dealer's Inquire**
- **Newsletter**
- **Rush Order**
- **Over 700 products**

Free Literature - Compare Products

Evaluate products Compare competitors. Learn about new alternatives. One *free call* brings information on just about any programming need. Ask for any ***"Packet"*** or "Addon Packet": ☐ ADA, Modula ☐ "AI" ☐ BASIC ☐ "C" ☐ COBOL ☐ Editors ☐ FORTH ☐ FORTRAN ☐ PASCAL ☐ UNIX/PC or ☐ Debuggers, Linkers, etc.

RECENT DISCOVERIES

SMALL TALK for PCDOS - "Methods" has objects, windows, browser, inspector. PCDOS $239

ARTIFICIAL INTELLIGENCE

EXSYS - Expert System building tool. Full RAM, Probability, Why, Intriguing, serious. PCDOS $275

GC LISP - "COMMON LISP", Help, tutorial, co-routines, compiled functions, thorough. PCDOS Call

INSIGHT 1 - Expert Sys. Dev't, decent PCDOS $95

M Prolog - full, rich, separate work spaces. MSDOS $725

PROLOG-86 - Learn fast, Standard, tutorials, samples of Natural Language, Exp. Sys. MSDOS $125

TLC LISP - "LISP-machine"-like, all RAM, classes, turtle graphics 8087. CP/M-86, MSDOS $235

Expert System front-ends for PROLOG: APES ($275), ES/P ($895)

Other solid alternatives include: MuLISP-86 ($189), WALTZ LISP for CPM ($159), MicroPROLOG ($275)

EDITORS FOR PROGRAMMING

BRIEF Programmer's Editor - undo, windows, reconfigurable, macro programs, powerful. PCDOS $195

VEDIT - well liked, macros, buffers, CPM-80-86, MSDOS, PCDOS $119

MACINTOSH

We evaluate, carry every available programmers product. Ask.

C LANGUAGE

C Terp by Gimbel, full K & R, .OBJ, ASM interface, 8087. MSDOS $275

INSTANT C - Interactive development - Edit, Source Debug, run. Edit to Run - 3 Secs. MSDOS $495

"INTRODUCING C" - Interactive C to learn fast. 500 page tutorial, examples, graphics. PCDOS $95

MEGAMAX C - native Macintosh has fast compile, tight code, K&R, toolkit, .OBJ, DisASM MAC $275

C ADDONS

APPLICATION TOOLKIT by Shaw - Complete: ISAM, Screen, Overlay mgnt, report gen, Strings, String math. Source. CPM, MSDOS $495

COMMUNICATIONS by Greenleaf ($159) or Software horizons ($139) includes Modem7, interrupts, etc. Source. Ask for Greenleaf demo.

C SHARP Realtime Toolkit - well supported, thorough, portable, objects, state sys. Source MANY $600

PRE-C - LINT, full, enhanced, all Cs. MSDOS $345

DEBUGGERS

PERISCOPE DEBUGGER - load after "bombs", symbolic, "Reset box", 2 Screen, own 16K. PCDOS $279

SOURCE PROBE by Atron for Lattice, MS C, Pascal. Windows single step, 2 screen, log file. $395

FORTRAN LANGUAGE

DR/Fortran-77 - full ANSI 77, 8087, overlay, full RAM, big arrays, complex NUMS., CPM86, MSDOS $249

MacFORTRAN - full '77, '66 option, toolbox, debugger, 128K or 512K, ASM-out option MAC $375

Ask about Microsoft, Supersoft, others.

OTHER LANGUAGES

ASSEMBLER - ask about FASM-86 ($95), ED/ASM ($100) - both are fast, compatible, or MASM ($125), improvements.

BetterBASIC all RAM, modules, structure. BASICA - like $185

HS/FORTH - '79 & '83 Standards, full RAM, ASM, BIOS, interrupts, graph, multi-task, optimizer MSDOS $250

MBP COBOL has screen control, strong doc, '74 interm., fast. MSDOS $680

SUPPORT PRODUCTS

BASIC DEVELOPMENT SYSTEM - (BDS) for BASICA; Adds Renum, crossref, compress. PCDOS $115

CODESMITH - visual, interactive debugger. Symbolize, modify code $129

FASTER C - Lattice users eliminate Link step. Normal 27 seconds. Faster C in 13 sec. MSDOS $95

PLINK-86 for Overlays, most lang., segment control. MSDOS $325

"C" LANGUAGE

	OUR PRICE
MSDOS: C86-8087, reliable	call
Instant C - Inter., fast, full	495
Lattice C - the standard	call
Microsoft C 3.0 - new	279
Williams, debugger, fast	call
CPM80: EcoPlus C - faster, SLR	275
BDS C - solid value	125
MACINTOSH: Hippo II	375
Megamax - optimizer, full	275
Consulair's MAC C	275

Compare, evaluate, consider other Cs

BASIC

	RUNS ON	
Active Trace-debug	86/80	75
BASCOM-86 - MicroSoft	8086	279
BASIC Dev't System	PCDOS	115
BetterBASIC - 640K	PCDOS	185
CB-86 - DRI	CPM86	419
Prof. BASIC Compiler	PCDOS	89
CADSAM-Full Btree, source	MSDOS	150
SCREEN SCULPTOR	PCDOS	115

Ask about ISAM, other addons for BASIC

SERVICE

ALL PRODUCTS - We carry 700 products for MSDOS, CP M 86, CP M 80, MacIntosh and key products for other micros.

EDITORS Programming

	RUNS ON	OUR PRICE
BRIEF - Intuitive, flexible	PCDOS	195
C Screen with source	86/80	75
Epsilon - like EMACS	PCDOS	195
FINAL WORD-for manuals	86/80	215
PMATE-powerful	8086	185
VEDIT-full, liked	86/80	119
XTC - multitasking	PCDOS	95

COBOL

Dig. Res-decent	MSDOS	500
Macintosh COBOL-Full	MAC	1850
MBP-Lev. II, native, screen	MSDOS	885
Micro Focus Prof.-Full	PCDOS	call
Microsoft-Lev II, no royal	MSDOS	500
Ryan McFarland-portable	MSDOS	695

LANGUAGE LIBRARIES

GRAPHICS: GraphiC-source in C	MSDOS	219
GRAPHMATIC-3D. FTN, PAS	PCDOS	125
HALO-fast, full-all lang.	PCDOS	139
FILE MGNT: BTrieve-all lang.	MSDOS	215
CIndex + -source, no royal.	86/80	369
CTree-source, no royal.	ALL	369
dBC ISAM by Lattice	8086	229
dB VISTA-"Network" Structure	MSDOS	465
PHACT-up under UNIX, addons	MSDOS	225
OTHER: CUtil by Essential	MSDOS	129
Greenleaf - 200 +	MSDOS	159
CSharp - **Real-Time**	MSDOS	600
PORTABLE C to PC, Mac, II	Many	125
SOFT Horizons - Blocks I	PCDOS	139
SCREEN: CURSES by Lattice	PCDOS	125
CView - input, validate	PCDOS	195
MetaWINDOW - icons, clip	PCDOS	139
PANEL - many lapg, term	MSDOS	249
ProScreen - windows, source	PCDOS	415
Windows for C	MSDOS	175

FORTRAN

	RUNS ON	OUR PRICE
MS FORTRAN-86 - Impr.	MSDOS	$ 239
DR Fortran-86 - full '77'	8086	249
PolyFORTRAN-XREF, Xtract	PCDOS	165

OTHER PRODUCTS

Assembler & Tools - DRI	8086	159
Atron Debugger for Lattice	PCDOS	395
cEnglish - dBase to C	MSDOS	750
C Helper: DIFF, xref, more	86/80	135
CODESMITH-86 - debug	PCDOS	139
MacASM-full, fast, tools	MAC	115
MBP Cobol-86 - fast	8086	885
Modula 2 for	MAC, PCDOS	90
Micro: SubMATH-FORTRAN full	86/80	250
Microsoft MASM-86	MSDOS	125
MSD Debugger	PCDOS	119
Multilink - Multitasking	PCDOS	265
PC/FORTH + -well liked	MSDOS	219
PFIX-86 Debugger	MSDOS	169
PL/1-86	8086	495
Polylibrarian - thorough	MSDOS	95
PolyMAKE	PCDOS	95
PROFILER by DWB - flexible	MSDOS	109
Prolog-86-Learn, Experiment	MSDOS	125
SYMD debugger-symbols	PCDOS	119
TRACE86 debugger ASM	MSDOS	115

Note: All prices subject to change without notice
Mention this ad. Some prices are specials
Ask about COD and **PO**s. All formats available

Call for a catalog, literature, and *solid value*

800-421-8006

THE PROGRAMMER'S SHOP™
128-L Rockland Street, Hanover, MA 02339
Visa Mass: 800-442-8070 or 617-826-7531 MasterCard 8517

CIRCLE 94 ON READER SERVICE CARD

PUBLIC DOMAIN SOFTWARE REVIEW

More Macintosh software

By Tim Parker

Last month we turned our attention to the Macintosh and some of the programs made available recently by the Public Domain Exchange. As promised, we will continue this month with a few more of the Mac's better public domain releases.

Let's start with a short look at a disk that, while not especially useful in terms of utility programs or enhancements (with one exception), sure is a lot of fun! Mac Disk 9 is labeled "Voice and Sound Synthesizer" and consists of eight programs of that genre.

The one that immediately captures the attention is a file called Talking Head, which draws a caricature of a face on the screen. Pushing the mouse's button causes the mouth on the face to move a little, and the speech synthesis capabilities of the Mac are demonstrated by a few well chosen sayings. Examples are Hamlet's "To be or not to be" soliloquy and Abraham Lincoln's "Four score and seven years ago" address. The synthesis is very well done, and there is enough of a variety of sayings that a good demonstration of the Mac's capabilities along this line can be made.

Speech Lab allows the user to enter text that is translated and spoken by the Macintosh. Straight text entry is not very effective, and a phenom-based approach works much better. The approach to entering text for this program reminded me very much of the Votrax synthesis device. With a little practice, any saying can be entered and enunciated by the Mac.

A series of files are used to demonstrate the four-tone synthesis capabilities of the Mac. Sample tunes include the inevitable "Star Wars Theme" (sorry, but I'll stick to the orchestrated version) and a demonstration on building a sound like a gun shot. Finally, a square wave music demonstration is available.

A very useful utility is a program called Set File, which displays the files on the disk with their attributes, including date, locked, invisible, bundle, system, type, creator, and more. These can be changed at will by the user. For finding invisible files or making files invisible, this is very useful.

Mac Disk 10 is an interesting volume. It contains five folders. Read Me First has information on copying the disk and putting MS-BASIC and Finder on it. The other four folders are Applications, Games, Demonstrations, and Math Font. The Applications folder consists of the communications program KERMIT, with a well-written documentation file printable with MacWrite.

KERMIT, for those unaware of it, is a communications system for mini-micro-mainframe systems to allow a reasonable level of compatibility between them. It defines several standards for protocols and is an attempt to create a more universal environment for communications. (KERMIT will be the subject of a future column, so will not be discussed in detail here.)

Rolodex Database is exactly what its name implies. It is a Macintosh Rolodex. (I thought that was a tradename, but no acknowledgment of such is given.) It has some well-written, brief instructions on using the system and allows "cards" to be added to a master file. Printouts are available easily, and files can be found on any word or phrase that occurs anywhere on the card. This seems ideal for many different purposes. A quick phone directory, business card list, and library application come easily to mind. Although not blindingly fast (what is on the Mac?), it runs at an acceptable rate.

Hands On is listed as a "binary tree playground." Perhaps not for everyone, this is a program that allows binary trees to be built up relatively easily and then manipulated. Binary trees are shaped like trees, with branches, roots, leaves, and junctions referred to as nodes in binary tree parlance.

Binary trees have long been the subject of academic curiosity. Virtually every programming course at a university or college includes some sort of binary tree application. This program allows them to be taken on a more friendly level. Documentation is good, and the implementation is very good. Parsing is by infix, prefix, or postfix. (For more on binary trees, see this issue's CrossThoughts column.)

The Games folder should have been called Game, as only one item is in it. That is a game called Iago, which closely resembles GO. It is a strategic capture game where an opponent's piece has to be surrounded. There are three difficulty levels (novice, hacker, and expert), with three choices of board sizes (8x8, 10x10, and 12x12). The graphics are extremely well done, and the game is quite absorbing. It can be played against the computer or against another human. The computer plays a very good game!

A cute feature of Iago borrowed from a game first seen on the Apple is a "Hide the boss is coming" menu choice. When this is activated, the screen is saved and cleared, and a blank spreadsheet form appears. The instructions wisely caution you to have an intense look of concentration on your face when this is displayed. Naturally, the game can be recalled easily.

Speaking of instructions, Iago has a fairly good set as the rules are quite simple. Games can be played back at varying speeds, and a number of other options are available. All in all, this was a very interesting game and will probably continue to occupy a great many idle hours (and some that are not supposed to be idle).

The Demonstrations folder has two files in it. 3-D Demo draws a Macintosh on the screen and rotates it. UI Demo illustrates the different user interfaces available on the Mac. Although the rotating Mac shows three-dimensional capabilities, the folder as a whole wasn't quite as exciting as I'd hoped. This folder also has a BASIC 3-D graphics program and a BASIC pattern and cursor editor.

Finally, and of some interest to scientists, engineers, and mathematicians, is a set of Math Fonts for MacWrite. These modify the 10- and 20-point Seattle fonts. Most foreign language characters have been reassigned to mathematical symbols, but the Greek character set remains intact. The set is rather comprehensive and should be of great help to most technical report writers. Included characters are (randomly picked): Planck's constant, integral sign, several braces, summation sign, set theory symbols, proportional sign, congruent, logic symbols, and many more. A documentation file lists the choices and three keyboards available.

Mac Disk 11 is full of communications and utility programs. They all require MS-BASIC to run. Two communications programs vie for attention.

Red Ryder (I don't know where the name comes from) is a terminal program that accepts mouse commands. Documentation is included in a text file but be warned . . . it is long! Two versions of Red Ryder are included on the disk. One is an executable version, while the second is an ASCII text version that can be transmitted.

The other communications program is MacAck, which also supports the mouse. It emulates a standard terminal and has good online help menus.

A folder called Catalogger allows a catalog of Mac disks to be built up. Four programs compose the folder. One is a documentation file, while the others allow the catalog to be built, listed, and updated. Catalogger worked well in practice and was easy to use. Searching for titles is possible with a dump to either the screen or printer.

An MS-BASIC folder contains a number of programs of varying utility. Files on Ext Drive allow a catalog of the disk in the Mac's external drive to be obtained. Get Info allows a user to do several things to a file's attributes, including hide it (make it invisible), change file types, or use a special series of Toolbox calls. A help file is incorporated.

Split File will bisect MS-BASIC program listings so that they can be read and printed out with MacWrite. Pattern Help determines the code for background fill. Similarly, Pattern Editor allows (you guessed it) the background pattern to be edited. MacCursor allows a custom cursor to be designed. Tamler Menus is an example of pull-down menu procedures using MS-BASIC. A version without the REM statements is included, but the REM statements have all the documentation in them.

Finally, Type allows a file to be dumped to the screen, while Print Bas Files and Print Text will send the files to a printer.

Mac Disk 14 contains a potpourri of material. A MacPaint folder has three items in it. Explosion, The First Night, and Ground Hog Day 1984 have to be seen, not described.

A list of fixes for MS-BASIC version 1.01 is included in a MacWrite document. These were downloaded from a CompuServe user, who made a list of the updates, fixes, and additions to MS-BASIC. This file may be of some use to those who purchased the older version and also points out some of the major points that have irritated BASIC users.

Version 1.87 of MacTEP is available. This program was mentioned last month, and this is an expanded, corrected version. A MacWrite document file is also included. Enhancements include printer on/off toggling, XON/XOFF protocol, and others. A subsidiary program, Runner, is included.

MacMon is a program that provides a subset of Apple Monitor commands. A documentation file is included and is fairly well written. Available commands are displaying memory, altering memory, machine code disassembly listing, and memory moves. Double precision hexadecimal arithmetic is available.

An MS-BASIC folder contains several diverse items. Briefly, BigPic will enlarge

GOOD NEWS!

C for the 6809 WAS NEVER BETTER!

INTROL-C/6809,Version 1.5

Introl's highly acclaimed 6809 C compilers and cross-compilers are now more powerful than ever!

We've incorporated a totally new 6809 Relocating Assembler, Linker and Loader. Initializer support has been added, leaving only bitfield-type structure members and doubles lacking from a 100% full K&R implementation. The Runtime Library has been expanded and the Library Manager is even more versatile and convenient to use. Best of all, compiled code is just as compact and fast-executing as ever - and even a bit more so! A compatible macro assembler, as well as source for the full Runtime Library, are available as extra-cost options.

Resident compilers are available under **Uniflex, Flex** and **OS9.**

Cross-compilers are available for **PDP-11/UNIX** and **IBM PC/PC DOS** hosts.

Trademarks:
Introl-C, Introl Corporation
Flex and Uniflex, Technical Systems Consultants
OS9, Microware Systems
PDP-11, Digital Equipment Corp.
UNIX, Bell Laboratories
IBM PC, International Business Machines

For further information, please call or write.

INTROL CORPORATION
647 W. Virginia St.
Milwaukee, WI 53204
(414) 276-2937

CIRCLE 32 ON READER SERVICE CARD

Add EDITING to your Software with CSE Run-Time™

Your program can include all or a portion of the *C Screen Editor* (CSE).

CSE includes all of the basics of full screen editing plus source in C for only $75. For only $100 more get CSE Run-Time to cover the first 50 copies that you distribute.

Use capabilities like Full cursor control, block move, insert, search/replace or others. Portability is high for OSes, terminals, and source code.

Call for the ''CSE Technical Description'' and for licensing terms and restrictions.

Full Refund if
not satisfied in
first 30 days.
Call 800-821-2492

Solution Systems
335- Washington Street
Norwell, MA 02061
617-659-1571

CIRCLE 21 ON READER SERVICE CARD

PROFESSIONAL PROGRAMMER'S BULLETIN:

Be Productive, Be BRIEF™

The Programmer's Editor

TRY BRIEF "RISK-FREE" FOR 30 DAYS WITH OUR MONEY-BACK GUARANTEE!

BRIEF's power and flexibility provide dramatic increases in programming productivity. BRIEF's ergonomically designed human interface becomes a natural extension of your mind, allowing you to eliminate tedium and concentrate on creativity.

- **WINDOWS**
- **Full UNDO (N Times)**
- **Compile within BRIEF**
- **Keystroke Macros**
- **Exit to DOS inside BRIEF**
- **Programmable Macro Language**
- **Multiple files, unlimited size**
- **"Regular Expression" search**
- **Reconfigure keyboard**
- **Language sensitive user controllable features (such as Auto-Indent for C)**

AVAILABLE FOR PC-DOS, IBM-AT, AND COMPATIBLE SYSTEMS
ONLY $195.

CALL TOLL FREE
800-821-2492
for "Technical Description" or to order.

Solution Systems™
335-L Washington St., Norwell, MA 02061
617-659-1571

BRIEF is a trademark of UnderWare.
Solution Systems is a trademark of Solution Systems

CIRCLE 95 ON READER SERVICE CARD

C Helper™

FIRST-AID FOR C PROGRAMS

Save time and frustration when analyzing and manipulating C programs. Use C HELPER's UNIX-like utilities which include:

DIFF and **CMP** – for "intelligent" file comparisons.
XREF – cross references variables by function and line.
C Flow Chart – shows what functions call each other.
C Beautifier – make source more regular and readable.
GREP – search for sophisticated patterns in text.

There are several other utilities that help with converting from one C compiler to another and with printing programs.

C Helper is written in portable C and includes both full source code and executable files for $135 for MS-DOS, IBM AT CPM-80 or CPM-86. Use VISA, Master Card or COD.

Call: **800-821-2492**

Solution Systems™
335 Washington Street
Norwell, MA 02061
617-659-1571

CIRCLE 96 ON READER SERVICE CARD

PROLOG-86™

Become Familiar in One Evening

Thorough tutorials are designed to help learn the PROLOG language quickly. The interactive PROLOG-86 Interpreter gives immediate feedback. In a few hours you will begin to feel comfortable with it. In a few days you are likely to know enough to modify some of the more sophisticated sample programs.

Sample Programs are Included like:

- **an EXPERT SYSTEM**
- **a NATURAL LANGUAGE INTERFACE** (it generates a dBASE II "DISPLAY" command)
- **a GAME** (it takes less than 1 page of PROLOG-86)

PROTOTYPE Ideas and Applications QUICKLY

1 or 2 pages of PROLOG is often equivalent to 10 or 15 pages in "C" or PASCAL. It is a different way of thinking.

Describe the FACTS and RULES without concern for what the computer will have to do. Maybe you will rewrite in another programming language when you are done.

Programming Experience is not required but a logical mind is. PROLOG-86 supports the de facto STANDARD established in "Programming in Prolog."

AVAILABILITY: PROLOG-86 runs on MSDOS, PCDOS, IBM AT or CPM-86 machines. We provide most formats. The price of PROLOG-86 is **only $125.**

Full Refund if not
satisfied during
first 30 days.
800-821-2492

Solution Systems™
335-L Washington Street
Norwell, MA 02061
617-659-1571

CIRCLE 97 ON READER SERVICE CARD

a MacPaint document to four times its original size and allows the picture to be saved in four sections. These can be printed separately and then pasted together. FontList provides a list of all the fonts in the system and their corresponding font numbers. When auxiliary fonts such as the previously mentioned math font are used, this provides a handy reference list.

XRef provides a BASIC cross-referencing facility for variables. Compare does a straight comparison of two text files and lists their differences. DskZap is an interesting program that allows the Macintosh internal disk drive to be read, edited, and written in 512-byte blocks. This program should be used by experienced programmers, as the disk can easily be trashed.

To round off this month's summary of Macintosh material, Mac Disk 17 contains several items of interest.

Those tired of waiting for the Macintosh's disk copying routine will welcome the Four Pass Copier program. As its name implies, this program allows a disk to be copied in only four passes (assuming a single drive, of course!). Those who tend to make frequent backups and are tired of the inevitable insert messages should try this one.

Another enhanced BASIC program lister also appears on this disk. This adds a title, the date, and allows some versatility in listing. Boldfacing can be included, if desired, and separate statements can be placed on separate lines.

Another program of a similar sort is List, which is an MS-BASIC program that will send any ASCII text file to the Imagewriter. An automatic indentation of 12 spaces is set, and the text is automatically paginated. The program appears to crash if any lines longer than 160 characters are used, but this isn't really a limitation.

A MacWrite document called Bulletin Boards not surprisingly contains a list of some bulletin boards that the Mac can access. All the numbers are in the New York area.

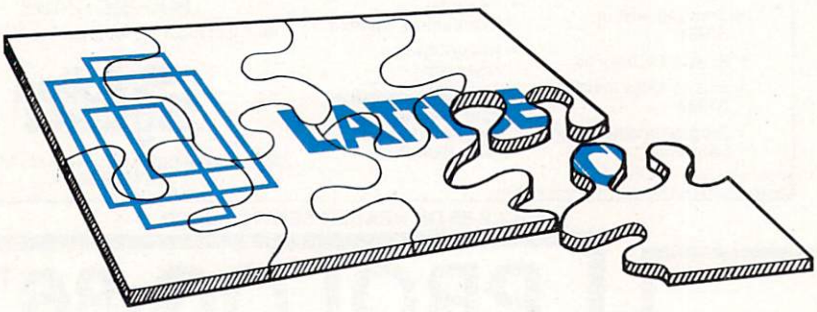
The graphics of the Mac are accessed by a few programs. Living Art displays a kinetic art demonstration. Exiting took a bit of trouble, as the cursor was hidden. I ended up moving the mouse everywhere until the pull-down menu was triggered. Perhaps that is half the fun?

A collection of different borders for incorporation into MacWrite and MacPaint documents is included in a Borders file. The sizes of the borders can be changed and rotated at whim. There is a good variety of different designs available, and they do tend to add that little extra touch to most documents.

Finally, MacPlot is a program designed for graphing. It is possible to create automatically scaled plots from several data sets at once. The graphs are well done, and the program is easy to use.

And so much for this brief look at the Macintosh. There is more to come in the future, especially as the amount of material available in the public domain is undergoing exponential growth. All of the mentioned disks are available from the Public Domain Exchange at 673 Hermitage Place, San Jose, Calif. 95134. The current price is $10 per volume. ■

Lattice makes the pieces fit . . .



Puzzled by complex programming problems?
Lattice has the tools to help you "get it together!"

Lattice provides the software tools and utilities you need to design, create, modify, analyze, and maintain your C programs and your documentation.

Lattice offers a wide range of C Compilers, C Cross-Compilers, C Function Libraries, and C Utility Programs to save you time and effort and help manage your complex software systems.

Lattice products are used by more than 26,000 software developers worldwide — including Lattice, Inc. as they develop their own new products, updates, and enhancements.

Call Lattice today. We have the programming tools you've been looking for to help you complete your projects!

Ask about our Trade Up to Lattice C Policy.

LATTICE®

P.O. Box 3072
Glen Ellyn, IL 60138
312/858-7950
TWX 910-291-2190

International Sales Offices
Belgium: Softshop. Phone: (32) 53-664875.
England: Round Hills. Phone: (0672) 54675.
Japan: Lifeboat Japan. Phone: (03) 293-2311.

CIRCLE 18 ON READER SERVICE CARD

PascalPac™ Not a Mayan Volcano But Packed with Power for Turbo Users.

(from the people who brought you tidy™—The Pascal Formatter)

For the IBM PC, XT, AT.

PascalPac™

X-REF creates cross reference table.
X-RAY browses cross reference and program simultaneously.
X-PRNT is a versatile listing program.
X-Peek browses programs or text files.
One version of PascalPac supports Microsoft and TURBO PASCAL.

tidy™

Program formatter makes a PASCAL program easier to read, understand and modify. **In use for over one year in major companies and programming organizations.** Available in separate versions for Microsoft and TURBO PASCAL.

"**tidy** is a lightening-fast Pascal formatter from Major Software."
"tidy does its job well and quickly."
"Of the products we've seen, we felt that tidy is the one most programmers will prefer."
PC Tech Journal

PascalPac **$195.00;** tidy-Turbo **$49.00;** tidy Microsoft **$69.00;** Shipping **$5.00.**

To order: VISA/MasterCard orders. **Call (415) 941-1924.** Or mail check/money order to: **Major Software, 66 Sylvian Way, Los Altos, CA 94022**

CIRCLE 16 ON READER SERVICE CARD

SuperSoft Programmer Utilities When Performance Counts

Star-Edit and Disk-Edit

Star-Edit is the professional programmer's text editor with an outstanding list of commands tailored to program development. It can greatly simplify all your editing tasks – moving and reproducing text or code, viewing two files simultaneously through separate windows, moving text or code between different files, searching forward or backward, and moving to the beginning or end of any word, sentence, paragraph, parentheses, or curly brackets. Virtual memory makes Star-Edit ideal for extremely large files; and because it never uses over 128K, it is well suited for multiple process and windowing environments. (PC DOS, MS DOS, CP/M-86, CP/M-80, UNIX, or XENIX): $225.00

To order call: **800-762-6629**
In Illinois call: **217-359-2112**
or write to **SuperSoft**.

Disk-Edit is the uniquely powerful disk utility for programmers which gives you access to every bit of information on your disk. It lets you read disk data in both HEX and ASCII, "text edit" any information on your disk, restructure disk information, and save lost or scrambled data. Imagine scrolling through your disk data, jumping between HEX and ASCII windows, and editing information anywhere on your disk. For all floppy and hard disk systems. (PC DOS, MS DOS, CP/M-86, CP/M-80, UNIX, or XENIX): $100

SuperSoft

1713 S. Neil St., P.O. Box 1628
Champaign, IL 61820
telex:270365

CIRCLE 79 ON READER SERVICE CARD

YOU NEED A GOOD C LIBRARY

POWER PACKS™

COMPLETE SOURCES NO ROYALTIES

COMPREHENSIVE C Power Packs include over 1000 functions which provide an integrated environment for developing your applications efficiently. "This is a beautifully documented, incredibly comprehensive set of C Function Libraries."
– Dr. Dobb's Journal, July 1984

USEFUL "...can be used as an excellent learning tool for beginning C Programmers..."
– PC User's Group of Colorado, Jan. 1985

FLEXIBLE Most Compilers and all Memory Models supported.

RECOMMENDED "I have no hesitation in recommending it to any programmer interested in producing more applications code, using more of the PC capabilities, in much less time." – Microsystems, Oct. 1984

■ **PACK 1: Building Blocks I** DOS, Keyboard, File, Printer, Video, Async	$149
■ **PACK 2: Database** B-Tree, Virtual Memory, Lists, Variable Records	$399
■ **PACK 3: Communications** Smartmodem™, Xon/Xoff, X-Modem, Modem-7	$149
■ **PACK 4: Building Blocks II** Dates, Textwindows, Menus, Data Compression, Graphics	$149
■ **PACK 5: Mathematics I** Log, Trig, Random, Std Deviation	$99
■ **PACK 6: Utilities I (EXE files)** Arc, Diff, Replace, Scan, Wipe	$99

Master Card/Visa, $7 Shipping, Mass. Sales Tax 5%

ASK FOR FREE DEMO DISKETTE

NOVUM ORGANUM INC.
SOFTWARE HORIZONS INC.

165 Bedford St., Burlington, MA 01803
(617) 273-4711

CIRCLE 25 ON READER SERVICE CARD

C Source Code

RED

Full Screen Text Editor

IBM PC, Kaypro, CP/M 80 and CP/M 68K systems.

- RED is fast! RED uses all of your terminal's special functions for best screen response. RED handles files as large as your disk automatically and quickly.
- RED is easy to use for writers or programmers. RED's commands are in plain English.
- RED comes with complete source code in standard C. RED has been ported to mainframes, minis and micros.
- RED comes with a Reference Card and a Reference Manual that provides everything you need to use RED immediately.
- RED is unconditionally guaranteed. If for any reason you are not satisfied with RED your money will be refunded promptly.

RED: $95

Manual: $10

Call or write today for
for more information:
Edward K. Ream
1850 Summit Avenue
Madison, WI 53705
(608) 231-2952

To order:

Either the BDS C compiler or the Aztec CII compiler is required for CP/M80 systems. Digital Research C compiler v1.1 is required for CP/M 68K systems. No compiler is required for IBM or Kaypro systems.

Specify both the machine desired (IBM, Kaypro or CP/M) and the disk format described (8 inch CP/M single density or exact type of 5¼ inch disk).

Send a check or money order for $95 ($105 U.S. for foreign orders). Sorry, I do NOT accept phone, credit card, or COD orders. Please do not send purchase orders unless a check is included. Your order will be mailed to you within one week.

Dealer inquiries invited.

NEW FOR THE MACINTOSH

PRESENTING THE MEGAMAX C COMPILER

FEATURING:
• IN-LINE ASSEMBLY • ONE PASS COMPILATION • SUPPORT OF DYNAMIC OVERLAYS • FULL ACCESS OF MACINTOSH TOOLBOX ROUTINES • AND MUCH MORE ...

DEVELOPMENT SYSTEM PACKAGE INCLUDES:
• FULL-SCALE IMPLEMENTATION (K&R) C COMPILER • THE STANDARD C LIBRARY • ROM ROUTINES LIBRARY • LINKER • LIBRARIAN AND DOCUMENTATION...

DEALER AND USER GROUP INQUIRES INVITED

$299.95

FOR MORE INFORMATION OR TO ORDER CALL OR WRITE:

Megamax, Inc.
BOX 851521, DEPT. 1
RICHARDSON, TX 75085-1521
(214) 987-4931

MACINTOSH IS A REGISTERED TRADEMARK OF APPLE COMPUTER INC

CIRCLE 13 ON READER SERVICE CARD

ConIX™

NOW ONLY $79.95!

If you think you're missing out on innovative software developments because nobody is writing for CP/M™-80, take a look at us. We've adapted UNIX™ features to CP/M like never before, and with the kind of professional, quality-controlled product that you deserve. That product is none other than the critically acclaimed ConIX Operating System.

ConIX can provide any 48K+ CP/M-80 or compatible system with I/O Redirection and Pipes (uses memory or disk), perfected User Areas, Command and Overlay Path Searching, Auto Screen Paging, 8Mb Print Buffering, 22 new SysCalls, Function Keys, "Virtual" disk system, Archiver (saves over 50% disk), extensive command language, 300+ variables, 100+ commands, pull-down menu, and much more! Uses as little as 1/2K RAM! Runs with CP/M for true data and software compatibility. Installs easily without any system mods!

The ConIX package lists at $165 and has been advertised and sold internationally to many enthusiastic customers since October 1983. As a special limited offer, we've lowered the price of the complete ConIX system by 50% to only $79.95! Don't miss this opportunity to bring your 8-bit micro back into the software revolution. Order your copy of ConIX today!

Price includes manual, 8" disk, and user support. 5¼" conversions available. Contact your local dealer, or buy direct and add shipping: $4.50 UPS, $10 Canada, $25 overseas. NY residents add sales tax.

Computer Helper Industries Inc.
P.O. Box 680 Parkchester Station, NY 10462
Tel. (212) 652-1786 (for information/orders)

"We're helping your computer work better for you!"

UNIX: AT&T Bell Labs, CP/M: Digital Research, ConIX: Computer Helper Ind.

CIRCLE 10 ON READER SERVICE CARD

Poor Person Software

Introduces

Write-Hand-Man

Desk accessories for CP/M

Write-Hand-Man lets you take notes, check phone numbers, make appointments, and countless other tasks without leaving Wordstar, dBase, Multiplan, or any other application. Enter **Write-Hand-Man** with a single keystroke and choose the program you want. When you leave **Write-Hand-Man,** your application continues normally.

$49.95 plus tax delivers **Write-Hand-Man** and 4 companion programs; **Notepad, Phonebook, Calendar,** and **Termcomm.** User written programs are easily added. All you need is M80 or some other LINK-80 compatible assembler.

Other CP/M products available from **Poor Person Software:** **Poor Person's Spooler** ($49.95), **Poor Person's Spelling Checker** ($29.95), **Poor Person's Spread Sheet** ($29.95), **Keyed Sequential Files** ($39.95), **Poor Person's Menus** ($29.95), **aMAZEing Game** ($29.95), **Window System** ($29.95), **Crossword Game** ($39.95), **Mailing Label Processor** ($29.95). Shipping included.

All products available on IBM 8 inch and Northstar 5 inch disks. Other 5 inch formats add $5 handling charge. No credit cards.

Poor Person Software
3721 Starr King Circle
Palo Alto, CA 94306
tel 415-493-3735

CP/M is a registered trademark of Digital Research

CIRCLE 67 ON READER SERVICE CARD

EXOTIC LANGUAGE OF THE MONTH CLUB

COMAL: A friendly micro language

By Mark R. Brown

COMAL (COMmon Algorithmic Language) is a powerful personal computer language that combines the best features of several popular languages.

A COMAL listing initially resembles Pascal due to the editor's automatic indentation of the control structures, which emphasizes the intrinsic Pascal-like structure of COMAL. On closer inspection, it reads like BASIC, because so many key words are identical or similar, as shown in the simple example in Listing 1. If a graphics procedure is examined, the code is a dialect of LOGO (Listing 2). These resemblances make COMAL instantly legible and easily learned by those who are already familiar with at least one of these languages.

For programming novices, COMAL may be the best of all possible languages to learn since it provides a structured programming environment, the instant reward of excellent graphics, and upward familiarity with so many other popular languages.

COMAL was originally developed by Borge Christensen in 1973 as a teaching language. His purpose was to retain BASIC's friendliness without its intrinsic sloppiness while providing Pascal's structured programming environment without its inflexibility.

The COMAL standards committee has established that COMAL be implemented as a kernel of standard commands augmented by extensions called packages. COMAL therefore retains a degree of transportability a level of magnitude higher than BASIC. A COMAL application written in kernel-level code should be fully transportable to other COMAL systems. The kernel contains the vocabulary necessary to perform all the standard I/O and processing functions.

Math operators are similar to those in BASIC and include some functions generally found only in extended versions of that language, such as MOD and DIV. Unfortunately, the floating point precision is the same as BASIC's, with no provision for double-precision math. Integer variables and arrays in the 16-bit range are allowed in decimal, hexadecimal, and binary notation. Logical operators are separated into Boolean and bitwise functions. For example, *A* AND *B* will return TRUE if *A* and *B* are both true, whereas *A* BITAND *B* returns the bitwise logical AND of the values *A* and *B*.

String functions resemble Pascal more than BASIC. Strings must be dimensioned before they can be used (for example, *DIM A$ OF 6*), though this is done automatically for input strings. Substrings are accessed by specifying subranges by position. For example, if *A$= "ABCDEF"*, then *A$(2:4)* is equal to *"BCD"*. The subrange may be reassigned as *A$(2:4):= "GHI"*, and *A$* becomes equal to *"AGHIEF"*.

Inclusion can be checked with the *IN* function. *"GHI"* in *A$* would return the value 2 since *GHI* is now a substring of *A$* starting at position 2. The BASIC function *LEN* is also available to determine the actual (not dimensioned) length of a string. All of the string operations may also be used on string array elements.

COMAL supports six common control structures. These have all been implemented previously in other systems, notably in Pascal and its progeny, though most languages do not offer them all.

The simplest of the structures is *REPEAT . . . UNTIL*, which provides a

```
0010 FOR x:=1 TO 10 DO
0020   FOR y:=1 to 10 DO
0030     PRINT x;"+";y;"=";x+y
0040     PRINT x;"-";y;"=";x-y
0050     PRINT x;"*";y;"=";x*y
0060     PRINT x;"/";y;"=";x/y
0070   ENDFOR y
0080 ENDFOR x
0090 END
```

Listing 1.

```
0010 PROC spiral_boxes
0020   setheading(0)
0030   FOR side:=1 to 24 DO
0040     box(50)
0050     left(15)
0060   ENDFOR side
0070   setheading(90)
0080   PROC box(length)
0090     FOR side:=1 to 4 DO
0100       forward(length)
0110       left(90)
0120     ENDFOR side
0130   ENDPROC box
0140 ENDPROC spiral_boxes
```

Listing 2.

loop with a single exit condition tested at the end of the loop. *WHILE . . . DO . . . ENDWHILE* is essentially the same thing, but the exit condition is tested at the start of the loop. Thus *REPEAT . . . UNTIL* is always executed at least once, but *WHILE . . . DO . . . ENDWHILE* will not be executed at all if the initial condition is not met.

LOOP . . . EXIT WHEN . . . ENDLOOP is a halfway version of this type of structure. The portion of the loop between *LOOP* and *EXIT WHEN* is executed every time through the loop, but the portion between *EXIT WHEN* and *ENDLOOP* is skipped after the exit condition has been met.

FOR . . . TO . . . STEP . . . DO . . . ENDFOR is a BASIC *FOR . . . NEXT* loop with a couple of additions. *STEP* is, of course, optional. The loop control variable is local, not global, thus eliminating possible conflicts with other variables in the program. If a command is included after the *DO* on the control line, the loop is interpreted as occupying only a single line. For example:

```
FOR X := 0 to 10 STEP 2 DO PRINT X
```

will print the even integers from 0 to 10. The *ENDFOR* is not needed to indicate the end of the loop, and any value *X* may have held before entering the loop is unchanged.

The *IF . . . THEN . . . ELIF . . . ELSE . . . ENDIF* structure has the same single-line format option for the simple *IF . . . THEN* case followed by a single operation. The multiline version allows *ELIF* and *ELSE* to continue control if the first condition tests false.

COMAL's most powerful control structure is *CASE . . . OF . . . WHEN . . . OTHERWISE . . . ENDCASE*. This allows multiple-choice type program flow, as in Listing 3. *OTHERWISE* is optional, but an error will occur if none of the cases specified are satisfied.

If you are a die-hard BASIC programmer and the idea of structured programming scares you, COMAL retains the *GOTO* statement, though its use is discouraged. It references not line numbers but LABELs.

Program listings automatically indent program structures, making it easy to follow program flow. The change from Pascal's ubiquitous *END;* for every structure to a separate syntax for each type of loop makes listings clearer, too. If line numbers bother you, the editor's *DISPLAY* command will list the program without numbers; they are used only by the editor and are never referenced by the program itself.

Three types of modules are used in COMAL. Error trapping is supported through the use of *TRAP . . . HANDLER . . . ENDTRAP*, which is similar to an *IF . . . ELSE . . . ENDIF* loop. The portion of code following *TRAP* is executed if no error condition exists. If an error occurs, the *HANDLER* code is executed.

System variables *ERR, ERRFILE,* and *ERRTEXT$* are used to determine the type of error encountered. *REPORT* can be used to define error conditions other than those detected by the operating system, if necessary.

The example in Listing 4 demonstrates a simple use of the *TRAP* module. In this example, the function will return any numeric value but will prompt the user with "Numbers Only" if nonnumeric data is entered.

Listing 4 also illustrates another COMAL module, the *FUNC*tion. These are defined and used as in Pascal. As is shown here, functions can be declared CLOSED, which makes all variables local. Individual variables may be be declared global by using IMPORT. A function may be numeric, array, or string, and always *RETURN*s a value.

The third module type is a *PROC*edure. This is similar to a function but does not return a value. Procedures can also be CLOSED and can use the IMPORT option. In fact, functions and procedures share all their characteristics and options with the sole difference that functions return a value.

Procedures and functions are allowed to be recursive in COMAL, as they are in most other languages with any real capability. The old standby factorial function serves to illustrate recursion in COMAL (Listing 5). You can see from this example that COMAL also allows nested functions and procedures. Once defined, functions are called by name. For example, *factorial(20)* would execute the function *factorial(n)*, returning the value for 20!. Procedures can be called by name or by using the optional *EXEC* (procedure).

COMAL procedures and functions may optionally be declared EXTERNAL. They must have previously been saved as disk files and will then be called from disk and overlaid on memory as needed. EXTERNAL procedures and functions must be CLOSED and may not use IMPORT. Listing 6 uses two external procedures, *add* and *subtract*.

The *add* procedure in Listing 7 is called from disk when needed and deleted from memory when done. If needed again, it must again be called from disk. While time-consuming, external procedures and functions allow much more flexibility when working in a limited memory space.

COMAL supports two sequential file types and a random file type. The standard ASCII file stores data as strings of characters. Each data element, numeric or string, is stored

```
0010 INPUT "Number 1-3":n
0020 CASE n OF
0030 WHEN 1
0040   PRINT"One"
0050 WHEN 2
0060   PRINT"Two"
0070 WHEN 3
0080   PRINT"Three"
0090 OTHERWISE
0100   PRINT"Out of range."
0110 ENDCASE
```

Listing 3.

```
0010 FUNC getnum(lin,pos,prompt$) CLOSED
0020   DIM n$ of 20
0030   LOOP
0040     INPUT AT lin,pos,12: prompt$: n$
0050     TRAP
0060       RETURN VAL(n$)
0070     HANDLER
0080       PRINT at lin,pos: "Numbers Only"
0090       FOR x:=0 to 1000 DO NULL
0100     ENDTRAP
0110   ENDLOOP
0120 ENDFUNC getnum
```

Listing 4.

in sequence followed by carriage return and line-feed characters as delimiters. A binary file is more compact, storing data as it is stored internally in machine memory. Floating point numbers are saved in 5 bytes and integers in 2. Strings are stored using a 2-byte character count followed by the character data.

Random files are more complex but allow freer access to data. The *CREATE (filename$,x,y)* command creates a random file with *x* number of records of length *y*. The number of records can be estimated initially and added to later, but the record length is inflexible once set.

COMAL also features BASIC style *READ* and *DATA* statements for imbedded data files. It always comes as somewhat of a shock to BASIC programmers when they find out that Pascal and many other languages do not have this capability, so it's reassuring to find it included here.

For want of a better label, let's talk about system "friendliness." The COMAL editor is a marvelous piece of work and makes writing programs a joy. Besides the listing conventions mentioned earlier, it features *AUTO* numbering and *RENUM*bering, *DEL*ete by range, *LIST* by procedure name, *FIND* and *CHANGE*, and options to list key words in uppercase or lowercase (though it makes no difference how you enter them initially). Control keys handle advance-by-word, abort edit, erase-to-end-of-line, and many other useful word processor-type functions.

COMAL also incorporates what is termed a three-pass run-time compiler. As a line is entered, syntax is checked. If an error is found, an error message is displayed and the cursor is placed on the error. Once corrections have been made, the error message disappears, and any text it may have overwritten is replaced.

The compiler even takes care of some simple mistakes and omissions automatically. For example, it will insert an omitted *DO, THEN,* or *OF,* and will convert *NEXT* to *ENDFOR*. This capability prevents a lot of headaches.

At run time, the compiler checks for and reports errors in program structure. It then compiles a name table in memory. While COMAL is an interpreted language, its structure and table usage make it an average of three to four times faster than BASIC and much faster than that in searches and sorts.

Once a program has been *RUN* (or *SCAN*ned, which executes the compiler without running the program), functions and procedures can be called from direct mode as though they were COMAL key words. This can be very useful and is a lot of fun with turtle graphics routines.

Variable and identifier names can be up to 78 characters long and can include the underline character as in Pascal, as well as apostrophes, brackets, etc.

Screen input and output are very friendly, too. *PRINT* includes the *AT* and *USING* options for screen formatting, and *ZONE* gives you control over the screen's tab positions. Input even has the *AT* option.

In addition, input may be restricted to only as many characters as you need (up to 120), and the cursor and screen clear keys are modified to work only within the context of the specified input window. This can be combined with error trapping to produce truly faultless input routines.

SETSCREEN and *GETSCREEN* allow the screen to be saved as a string variable, including cursor position. In this manner, a help screen or menu can be referenced and the screen easily restored when done.

COMAL features one useful capability borrowed not from BASIC, Pascal, or LOGO, but from CP/M: batch files. Batch files are disk text files that are treated by the system as though they were typed in directly from the keyboard. They are useful for establishing initial system parameters, linking custom character fonts, copying files, or other repetitive tasks that would otherwise require operator intervention.

Packages provide a means for extending the COMAL kernel. They tend to be more system dependent. For example, the Commodore 64 version of COMAL includes a SOUND package that provides access to its excellent SID synthesizer chip. (JOYSTICKS, PADDLES, LIGHTPEN, SPRITES, and character FONTs are also supported by the Commodore 64.)

Packages are written in machine code, *LINK*ed to the COMAL kernel, and called with the *USE* command, as in *USE SOUND*. *USE* links the key word vocabulary of the package to the COMAL inter-

```
0010 FUNC factorial(n) CLOSED
0020   TRAP
0030     RETURN fac(n)
0040   HANDLER
0050     RETURN 0
0060   ENDTRAP
0070   FUNC fac(n)
0080     IF n=0 THEN
0090       RETURN 1
0100     ELSE
0110       RETURN fac(n-1)*n
0120     ENDIF
0130   ENDFUNC fac
0140 ENDFUNC factorial
```

Listing 5.

```
0010 DIM reply$ of 1
0020 LOOP
0030   PAGE
0040   PRINT "a - add"
0050   PRINT "s - subtract"
0060   PRINT "x - exit"
0070   REPEAT
0080     INPUT AT 5,1,1: "Your choice : ": reply$
0090   UNTIL reply$>"" AND THEN reply$ IN "ASXasx"
0100   CASE reply$ OF
0110   WHEN "a","A"
0120     add
0130   WHEN "s","S"
0140     subtract
0150   OTHERWISE
0160     EXIT
0170   ENDCASE
0180 ENDLOOP
0190 END "End of example."
0200
0210 PROC add EXTERNAL "0:ext.add"
0220 PROC subtract EXTERNAL "0:ext.sub"
```

Listing 6.

preter. *DISCARD* is used to disconnect a package after it has been called. Packages can be called even in direct mode and their commands used from the keyboard.

Graphics are implemented as a package in COMAL. This means they will vary somewhat from system to system, though the standards committee recommends LOGO-style turtle graphics. The Commodore 64 system actually provides two mutually compatible graphics systems, Cartesian point-plot and turtle.

The turtle is by far the most fun to use and the most rewarding for the beginner. The turtle can be manipulated interactively in *SPLITSCREEN* mode and can include prescanned turtle graphics procedures for drawing complex objects. LOGO shorthand is supported, so *FORWARD(50)* can be entered as *FD(50)* to save keystrokes. This package also features the only windowing capabilities I've seen so far for the Commodore 64, providing for mixed graphics and text windows, screen-wrap options, and coordinate system scaling.

The extensibility of COMAL through the use of packages is one of its most appealing features. Packages can be linked to the kernel when needed and discarded when not needed to free memory space and speed execution time. They provide a means to access machine-specific capabilities with high-level commands while still providing a cross-system compatible language kernel. They allow the addition of vocabulary to handle unforeseen future needs, such as when a new piece of hardware is added to a system. Packages can even allow for error messages to be provided in the native tongue of the operator.

Where can COMAL be found? Well, it's starting to catch on, mostly in Europe, though its U.S. following is strong and growing stronger. IBM's recent announcement of COMAL for the PC is sure to add a boost to the movement. Apple is also reportedly working on a version (COMAL is available in Europe for Apple CP/M).

As a personal computer language, it's hard to beat. It has the capability to handle everything from games to serious applications and provides this in an environment that is structured and very fast for an interactive, interpreted language. If you have been looking for an alternative to BASIC or Pascal, or are involved in teaching programming, COMAL may be just what you need. ■

References

Lindsay, Len. *The COMAL Handbook*. 2nd ed. Reston Publishing Co., 1984.

COMAL Today Newsletter. COMAL Users Group USA Ltd., 5501 Groveland Terrace, Madison, Wisc. 53716

Mark Brown is a hacker and language junkie who has been involved with personal computing since day one. He is a freelance writer and programmer and associate editor of INFO 64, *a magazine devoted to the Commodore 64.*

```
0010 PROC add external proc CLOSED
0020   missed:=FALSE; count:=0
0030   WHILE NOT missed DO
0040     count:+1
0050     numb1:=RND(1,9)
0060     numb2:=RND(1,9)
0070     PRINT count,"> what is";numb1;
0080     PRINT "plus";numb2;
0090     INPUT "--->": reply;
0100     IF reply=numb1+numb2 THEN
0110       PRINT "yes"
0120     ELSE
0130       PRINT "oops"
0140       missed:=TRUE
0150     ENDIF
0160   ENDWHILE
0170   PRINT "answers were right,";
0180   PRINT "up to problem";count
0190 ENDPROC add external proc
```

Listing 7.

C
POWER
C LIBRARIES
C WINDOWS
Best You Can Get!
325 Fully Tested Functions
SIX C LIBRARIES
FUNCTIONS YOU DON'T HAVE BUT NEED!
All Source Code. No royalties.

51	screen handling/graphic functions	$49.95
50	cursor/keyboard/data input functions	$39.95
85	superior string functions	$59.95
25	system status & control functions	$29.95
72	utility/DOS/BIOS/time/date functions	$49.95
42	printer control functions	$29.95

RICHLY COMMENTED
EASY TO LEARN
EASY TO MODIFY
NO MATTER WHAT ELSE YOU HAVE
GET THESE!!
ANY 3 LIBRARIES $69.95
ALL 6 LIBRARIES $99.95
50 MOST NEEDED FUNCTIONS
$49.95
POWER WINDOWS
PROFESSIONAL WINDOW MANAGEMENT
OVERLAYS, BORDERS,
POPUP MENUS, HELP WINDOWS,
STATUS-LINE, COLOR HIGHLIGHTING,
AND MORE!!!
C WINDOWS: COMPLETE SOURCE CODE $99.95
ALL LIBRARIES PLUS WINDOWS $149.95
Entelekon
SOFTWARE SYSTEMS
ENTELEKON 12118 KIMBERLEY
HOUSTON, TX. 77024 (713)-468-4412
VISA•MASTERCARD•CHECK
CIRCLE 50 ON READER SERVICE CARD

6 TIMES FASTER!

SuperFast Software Development Tools

INCREASE YOUR PROGRAMMING EFFICIENCY
with high-performance software development products from SLR Systems.
No other tools approach the speed or flexibility of the SLR Systems line.

"Z80ASM is an extraordinary product...",
Robert Blum, Sept. 84 DDJ

"...in two words, I'd say speed & flexibility",
Edward Joyce, Nov. 84 Microcomputing

ASSEMBLERS

- RMAC/M80 macros
- Nested INCLUDES & conditionals
- 16 char. labels on externals
- Built in cross-reference
- Optional case significance
- Phase/dephase
- Math on external words and bytes
- Define symbols from console
- Generate COM, HEX, SLR-REL, or Microsoft-REL files
- Time & Date in listing
- Over 30 configure options

	Z80ASM -full Zilog Z80	$125
NEW!	Z80ASM+ -all tables virtual	$195
NEW!	SLRMAC -full Intel 8080, with Z80.LIB extensions internal	$125
NEW!	SLRMAC+ -all tables virtual	$195

Z80 CPU, CP/M compatible, 32K TPA required

"Z80ASM...a breath of fresh air...",
Computer Language, Feb. 85

MasterCard VISA C.O.D., Check or Money Order Accepted

LINKERS

- Links SLR & M80 format files
- Output HEX or COM file
- Three separate address spaces
- Load map and SID/ZSID .SYM file

SLRNK+ includes:

- All tables overflow to disk
- HEX files do not fill unused space
- Intermodule cross-reference
- EIGHT separate address spaces
- Works with FORTRAN & BASIC
- Generate PRL & SPR files
- Supports manual overlays
- Full 64K output

	SLRNK -fastest memory based	$125
NEW!	SLRNK+ -full featured virtual	$195

Combo Paks available from $199. - $299.

For additional information contact SLR Systems
1-800-833-3061, in PA (412) 282-0864
1622 N. Main St., Butler, PA 16001 • Telex 559215

SLR Systems

CIRCLE 73 ON READER SERVICE CARD

WIZARD C

Fast compiles, fast code and great diagnostics make Wizard C unbeatable on MSDOS. Discover the powers of Wizard C:

- ALL UNIX SYSTEM III LANGUAGE FEATURES.
- UP TO A MEGABYTE OF CODE OR DATA.
- SUPPORT FOR 8087 AND 80186.
- FULL LIBRARY SOURCE CODE, OVER 200 FUNCTIONS.
- CROSS-FILE CHECKS OF PARAMETER PASSING.
- USES MSDOS LINK OR PLINK-86.
- CAN CALL OR BE CALLED BY PASCAL ROUTINES.
- IN-LINE ASSEMBLY LANGUAGE.
- 240 PAGE MANUAL WITH INDEX.
- NO LICENSE FEE FOR COMPILED PROGRAMS.

The new standard for C Compilers on MSDOS!

Only $450

WSS

For more information call (617) 641-2379
Wizard Systems Software, Inc.
11 Willow Ct., Arlington, MA 02174
Visa/Mastercard accepted

CIRCLE 86 ON READER SERVICE CARD

PROLOG V

ONLY
$69^95

Examine the documentation for 30 days and return with disk still sealed for full refund, if not fully satisfied.

PHONE ORDERS:
(619) 483-8513

The Language of Artificial Intelligence

Full Prolog as defined in Clocksin & Mellish, *Programming in Prolog*, Springer-Verlag, Berlin Heidelberg New York, 1981.

User's manual with extensive tutorial, complete reference guide, and program examples. Custom-designed binder and slip cover.

PC-DOS/MS-DOS, 128K RAM

☐ PAYMENT ENCLOSED $ ______
Shipping and Handling Add: US $5, CAN $7.50
CA residents add 6% sales tax
☐ CHARGE MY: ☐ MasterCard ☐ Visa
Card No. ______ Exp. Date ______
Signature ______
Mr./Mrs./Ms. ______ *(please print full name)*
Address ______
City/State/Zip ______
CL

CHALCEDONY SOFTWARE

5580 LA JOLLA BLVD.
SUITE 126 G
LA JOLLA, CA
92037

CIRCLE 43 ON READER SERVICE CARD

You know that choosing the right software is serious business. So does WATCOM.

So before you make any decisions about your software needs, talk to WATCOM — the people major software users around the world have trusted for years.

WATCOM has the products you need to get the job done right. Proven performers like WATFOR*, WATFIV*, WATBOL*, and SCRIPT. Plus new leaders in software for PC workstations and micro-to-mainframe communications. Networks, language interpreters and compilers. Text preparation and data management.

All WATCOM products are human engineered to provide the optimum in people efficiency and productivity. And they're designed to run compatibly on IBM mainframes and PC's, Digital mainframes and micros, and Commodore micros.

Whatever you need is backed up by WATCOM's innovative maintenance and support services. You'll be kept up to date with the latest in product enhancements and information. And our publications and seminars will help you get the most out of your software investment.

WATCOM. Quality products. Professional service. And a reputation built on more than 150,000 licensed mainframe and micro software programs throughout the world. So talk to us before you decide. After all, choosing the right software is serious business. For you. And for WATCOM.

Make the right choice: *WATCOM INTERPRETERS*

Excellent error diagnostics make WATCOM Interpreters the right choice in software for efficient program development in APL, BASIC, COBOL, FORTRAN, or Pascal.

WATCOM Interpreters emphasize error detection so that program corrections are more easily executed. Hard-to-find errors can be quickly located with the integrated debugging system for COBOL, FORTRAN, and Pascal. And programs can be efficiently entered and corrected with the integrated full-screen editor in all languages but APL.

WATCOM Interpreters are available for IBM PC, IBM 370 VM/SP CMS, and Digital VAX VMS*.

Make the right choice. Call or write WATCOM today and we'll tell you all about WATCOM Interpreters or any of WATCOM'S other people-efficient products.

WATCOM

The right choice in software.

Yes! I want to make the right choice in software. Send me more information on: ☐ WATCOM INTERPRETERS ☐ WATCOM Software Catalogue

Name:________________________________

Company:________________________________

Title:________________________________

Address:________________________________

City:________________ **State:**____________ **Zip:**__________

WATCOM PRODUCTS INC.
415 Phillip Street
Waterloo, Ontario, Canada
N2L 3X2
(519) 886-3700
Telex 06-955458

*WATFOR, WATFIV and WATBOL are registered trademarks of the University of Waterloo.
*IBM PC and IBM 370 VM/SP CMS are registered trademarks of International Business Machines Corporation.
*VAX, VMS are registered trademarks of Digital Equipment Corporation.

CIRCLE 24 ON READER SERVICE CARD

On-line with Ward Christensen

By Craig LaGrow

The name Ward Christensen has become synonymous with microcomputer telecommunications and that widespread phenomenon called bulletin board systems. Now an IBM systems engineer living in the Chicago, Ill., area, Christensen looks back on the early days of micro communications and has some interesting tales to tell.

As a self-proclaimed hacker, he prides himself in his ability to be a technical troubleshooter—both in his work and in his hobbying. "I enjoy solving problems," says Christensen, "and to me true hacking comes from the phrase, 'can you hack it?'"

"I guess I'm a hobbyist partly because, in a sense, I have no goals," he says. "I think that's part of what differentiates a person from hobbying and commercial efforts. I don't have this grandiose plan of being a millionaire, or of wanting to set the world on fire with some program, or just wanting to get organized, sell, and advertise something. I've always just enjoyed twiddling with electronics and things like that, seeing what I could make them do."

Christensen recalls one of his earliest hacking challenges—the day he decided he wanted to build an auto-dialer for his phone. He imagined it would be a combination of a file on disk containing a way of sending data across a phone line over and over again. "I imagined I could try setting up something like a relay that would open the line and pulse it to dial the number," he remembers. "Well, yeah, I can hack that! I can do the software and the hardware and whatever I need to get the job done."

In college, Christensen spent one year as a chemistry major; but one day he decided he didn't like college at all and dropped out. Instead he got a job tinkering with computers, which were very new at the time and not very powerful. With his new-found interest and talent, he soon was inspired to go back to college, all in the hopes of landing a job with IBM after graduation—which he was finally able to do three years later.

"Maybe I wasn't smart enough to go looking for research or development jobs, which I probably would've liked better," he says. "I just took an interview and wound up being thrown in the hat in Chicago and being picked by the Hammond, Ind., branch for a job as a systems engineer—which is the technical support of sales, configuring and performance and things like that."

Christensen worked with literally every level of IBM computer equipment and has only recently been assigned to work on technical support for the IBM PC within the company. "I have avoided microcomputers within IBM like the plague because hobby is hobby and work is work, and never the two shall meet," he says.

Nevertheless, one day his boss showed up at his door saying, "We're tired of you helping everybody else in the area of PCs. How about helping IBM for a change?"

Christensen now works in a Chicago-based IBM sales office as a technical troubleshooter, answering questions from large corporate customers like, "How do I get the Quiet Writer to work with my Display Write II?" He does customized system configurations, like networking PCs with an AT, for example. He also is in charge of putting on customer seminars and demonstrations for his office.

Before the IBM PC came on the market, the big operating system for microcomputers was CP/M. In the CP/M programmer's community, Ward Christensen is today regarded as one of the founding fathers of public domain software. He is especially recognized for his MODEM telecommunications program, which is best known today by a variety of names such as MODEM7.COM, MDMxxx, and XMODEM.COM.

But now Christensen has shifted his programming interests to the IBM PC and away from CP/M. "I decided one day that I just wanted something slower, with a lousy keyboard," he jokes.

He says he made this change partly out of frustration. "But if you have to put it in a word, I'd say 'ego'. I enjoy being knowledgeable on things. I don't like being asked questions that I can't answer all the time. People started coming up to me with IBM PC questions a lot, both within and outside IBM, and I discovered that I wasn't the expert. I didn't even have a PC then."

"I don't know if I made this transition to the PC because CP/M was dead or dying or whatever. As a matter of fact, at that time there were certainly more CP/M machines out there than PCs, and there was also much more software available for CP/M. But that won't be for long.

"Outside of CP/M, there are some very fundamental things that are nice, like tree-structured directories and date- and time-stamped files. Secondly, I think the PC's strongest suit is the fact that it is a standard. The S-100 was a standard and fostered the growth of the microcomputer industry by having a standard place to plug your boards. If you wanted to make a who-knows-what board, there was a standard bus to plug it into.

"But the trouble was that there was no one company that was strong enough to drag the industry kicking and screaming into some higher level standards like a graphics board or communications interface. So the PC came along and, disregarding the better-or-worse, faster-or-slower issues, provided a standard interface for color graphics, a standard interface for communications, and large amounts of memory.

"It was standard. So if someone wanted to come out with something like a good graphics board, it simply couldn't exist in the S-100 world because no single board maker could make enough of any one kind for a software house to have any prospects of marketing any reasonable quantities of their package.

"Communication was different because people simply realized that you had to go through the effort of installing it for whatever modem and board you had. But, with the PC and communications, all you had to do was just drop it in. Maybe it was COM1: or COM2:, it's not is it this bit or that bit, is it ready when it's high or low? Plus you didn't have interrupt-driven things and so on. And now for graphics it's easy, now there's a big enough standard marketplace and a well-known graphics board that people can make things like Auto Cads."

In the mid-1970s, microcomputer hacking was just beginning. New ideas were being developed by hobbyists across the country. "Back then, people had the benefit of just knowing the kinds of things that needed to be hacked," Christensen remembers. "If we're talking about public domain programs, there are some standard programs like the library utilities and the squeeze and unsqueeze utilities that will exist no matter what the machine is. They are just fundamentally nice things to have around."

"To me, the most difficult thing today is to find something to invent that hasn't already been done, and I had the supreme luxury of programming in a void."

Christensen and Randy Suess's system, CBBS/Chicago, is still running in Suess's basement. Christensen also has his own "test CBBS" running in his basement. The CBBS program they wrote is no less than 14,000 lines of 8080 assembler code and assembles into a 27K executable file.

In terms of the language he likes best to program within, Christensen is definitely biased toward the assembly language level. "We really don't need another high- or low-level language—we need an intermediate-level language. We need a language that understands the hardware just a little bit better, even better than C. I often see some pretty awful code which gets generated by C."

But when Christensen originally wrote his CBBS program in 8080 assembler code, his intention was ironically *not* to create a system for file transfers. In fact, he never really did have a file transfer system up—except for very early on—and that was only for a little while. "I've always been more interested in what people have to say message-wise," he says.

"In the early days, I remember putting up a message on my bulletin board saying, 'Who put up the first remote CP/M system?' and the answer kept coming back, 'You did!' Apparently I was the first person to put a system on-line that was not my only system," he says.

But the real problem with file transfers at the time, notes Christensen, were the low-storage 70K disks. "People like Keith Petersen just kept calling in all the time and filling them! And I wasn't into wanting to see a bunch of code from other people. I had so many things that I was working on of my own that I took it back down after a fairly short while. But, supposedly, that was the very first dial-in CP/M system."

In the beginning, CBBS was for anybody calling in with a dumb terminal. People with teletypes and built-in modems would often take breaks by calling into Christensen's system from their office terminals at work. "It makes sense that CBBS wasn't set up as a file transfer system because most of the people with modems at the time didn't necessarily have computers," he says.

Christensen set up CBBS with prompt bells programmed into the system so that every time it asked you to respond to a question like "What's your first name?," "Function?," "Enter Starting Message Number," etc., the system would go "BEEP!, BEEP!, BEEP!" This infuriated callers who left messages saying, "Can't you kill the bells?"

"Aha! I know what's going on—the back rooms of industry are calling up CBBS during business hours, and they're being embarrassed by all the beeps," says Christensen. "So I put in a *P* command that allowed callers to turn off the prompting bell. Immediately after that, I got back comments like, 'The back rooms of industry thank you!'"

Christensen first came in contact with acoustic modems back in the early 1970s when they were used mainly for terminals

New Release

C + UTILITY LIBRARY = PRODUCT

- We have over 300 complete, tested, and, documented functions. All source code and demo programs are included.
- The library was specifically designed for software development on the IBM PC, XT, AT and compatibles. There are *no royalties.*
- Over 95% of the source code is written in C. *Experienced programmers* can easily "customize" functions. *Novices* can learn from the thorough comments.

We already have the functions you are about to write

Concentrate on software development—not writing functions.

THE C UTILITY LIBRARY includes:
- Best Screen Handling Available
- Windows
- Full Set of Color Graphics Functions
- Better String Handling Than Basic
- DOS Directory and File Management
- Execute Programs, DOS Commands and Batch Files
- Complete Keyboard Control
- Extensive Time Date Processing
- Polled ASYNC Communications
- General DOS BIOS gate
- Data Entry
- And More

- The Library is compatible with: Lattice, Microsoft, Computer Innovations, Mark Williams and DeSmet. Available Soon: Digital Research, Aztec and Wizard.

C Compilers: Lattice C—$349, Computer Innovations C86—$329; Mark Williams C—$449.

C UTILITY LIBRARY $185. Special prices on library & compiler packages.

Order direct or through your dealer. *Specify compiler* when ordering. Add $4.00 shipping for UPS ground, $7.00 for UPS 2-day service. NJ residents add 6% sales tax. Master Card, Visa, check or P.O.

ESI
ESSENTIAL SOFTWARE, INC
P.O. Box 1003 Maplewood, New Jersey 07040 914 762-6605

CIRCLE 33 ON READER SERVICE CARD

dialing into mainframes. "Back then they had a nice-looking wooden box about 5-in. wide and maybe 10-in. long and a foot high," he says. "And you would set the phone inside this cushioned, foam-protected and elegantly styled wooden box."

He bought his first modem from a company called Livermore in 1974. This was the same brand of acoustic coupler that IBM was using internally at the time. "I really don't know how much before that time I actually started using them, along with a dial-up 2741 selectric terminal from work," he says.

One problem with the Livermore modem, however, was that it was an "originate only" modem. But the packing materials for this modem also talked about an "originate/answer" modem that Christensen realized must be the type he'd require to have two modems talk to each other, so he upgraded it immediately. This turned out to be a fortuitous decision, made three years before the invention of his file transfer protocol, which required one of the modems to be in answer mode.

Christensen describes the invention of the bulletin board software concept: "If you've been to club meetings, you know how a cork bulletin board can play an important part—people offering to sell things, offers for group purchases, bad comments about products not recommended to buy, etc. And I thought, why not just mechanize or computerize a bulletin board. That's how the name came about too."

Between Christensen and his friend Randy Suess the term Computerized Bulletin Board System (CBBS) was chosen for their new invention. "I really don't know if I invented the term or if Randy did. I have a hunch I did; but I have a very bad memory, and I hate to take credit for something I'm not sure of," says Christensen. "I don't know if we hadn't actually invented a BBS, whether somebody else would've or not."

Christensen remembers that one night Suess was doing some pistol target practice in the basement for fun—shooting at a target they had taped up on a wall, not realizing that the power cord for a wall clock lay behind the target. "With one shot, Randy managed to neatly sever the cord, and in doing so, blew a fuse and took the CBBS down," he says.

"So the next day, as I called into CBBS to check for mail, a fellow had left me a message saying how the system had suddenly froze on him and he was worried whether he had done something wrong to cause this. My reply was simply, 'No it wasn't you, Randy shot it!'"

Christensen began working on the first version of his MODEM.COM program soon after he bought his first modem—the same time he was working on some limited communications programming with large IBM systems. He also was shopping for a home computer.

"I had manuals for the Data General NOVA and the TI 980A minis and things like that," he remembers. "For $10,000 you'd get a couple cassettes and a thermal printer and 16K, and that wasn't exactly right."

In 1974, he was trying to build his own TTL-based machine at home. He often spent his evenings breadboarding things and oscilloscoping them, block diagramming, etc., with the intention of "getting a computer I could program."

Yet when the January 1975 issue of *Popular Electronics* came out, the cover showed the new Altair microcomputer. "I said to myself, 'Now I don't have to build my own!'"

During this time period he built things like hardware random number generators, shift registers, and little push button start-and-stop counters. Christensen even breadboarded the components for a miniature computer of his own.

"But the Altair was really my saving grace," he admits. "I now could actually buy a machine and program it and not have to design and build one myself. While I sat impatiently waiting for my

PROGRAMMING TOOLS
for Pascal and C

C Productivity Series—
The Professional's Edge

Blaise Computing has a range of programming aids for the most popular C compilers in the IBM environment that no serious system developer should be without. These packages help you to easily access advanced capabilities of the hardware and operating system, and to finish your projects with a substantial saving of time and effort. With software development costs and pressures as great as they are, can you afford not to take advantage of the finest tools available?

- **C TOOLS**™ puts advanced string handling functions at your disposal and provides a high-level interface to all BIOS functions from your C program. Complete screen handling, graphics primitives, and a substantial group of useful, general-purpose functions are also featured. **$125**
- **C TOOLS 2**™ lets your program perform all the advanced DOS 2.0 services. Program chaining, software interrupt handling, and dynamic memory allocation are all done "right." Buffer and file handling functions are provided, as well as a general DOS gate. **$100**
- **C VIEW MANAGER**™ is our display screen management system that makes screen development and documentation much faster. It comes with a complete library of C functions which use the screens you have developed to recall and display information, capture and validate field data entry, and provide context-relevant help files. **$275**
- **ASYNCH MANAGER**™ is a library of interrupt-driven routines providing a general interface to both COM ports for your asynchronous communications applications. Introductory price of **$175** includes all source.

All of these products may be used by developers with no royalty payments to Blaise Computing. Source code either comes with the package, or is available. We support Lattice, Computer Innovations, and Microsoft C compilers. To expedite your order or to obtain further information, call or write us directly.

Blaise Computing's Programmer Productivity series is also available in versions for the Pascal language.

◆

BLAISE COMPUTING INC.
2034 Blake Street Berkeley, CA 94704
(415) 540-5441

CIRCLE 8 ON READER SERVICE CARD

INTERACTIVE PASCAL
Introductory Offer
Now Only $39.95

Announcing MYSTIC PASCAL—the interactive Pascal system. Command statements are INSTANTLY compiled and executed. As easy to use as Basic—AND its a true object code compiler!

- MYSTIC PASCAL is interactive—you can instantly compile and execute statements
- True 8088/8086 object code compiler
- We won't say how fast the compiler is, because you absolutely wouldn't believe it!
- Full screen editor for Standard Pascal programs
- Pop-Up Help windows describe the Standard Pascal language
- True MULTI-TASKING support
- Satisfaction guaranteed! 15 Day refund with no questions asked!

MYSTIC CANYON SOFTWARE
P.O. Box 1010
Pecos, New Mexico 87552

Place your order today!
Phone or use the coupon!
(505) 988-4214
10 to 5 Mountain Time

Name ____________
Address ____________
City ____________
State ________ Zip ________

Price is $39.95 plus $4 shipping. Outside US & Canada add $20 shipping. Payment must be in US funds on a US bank. Purchase orders accepted from recognized institutions.
NM residents add sales tax.

☐ Check/MO ☐ COD ☐ VISA ☐ MC

Card ____________
Exp. ____________

Altair, I shopped through some magazines to get a disk drive system."

Knowing his eventual goal was a computer with floppy disks, Christensen found and bought some dual-disk drives which stored 256K on a floppy disk stored in a rigid plastic case and read with a flying head similar to the Bernoulli drives of today.

"So I bought those drives, got my Altair, built my own interface to a selectric typewriter, constructed my interface to the drives, and got a hold of a little assembler/editor from Processor Technology—4K worth of programs that I received on paper tape. And I think I even typed in the hex for it. After that I wrote a little hex loader. That's how I got a first little assembler/editor going. I used their video display module, or VDM, which really opened up the computer to usefulness.

"So eventually I got to where I could load and save files and programs and do some assemblies on this kludge of hardware and software that really was not running under any operating system per se. I adopted Processor Technology's 4K editor/assembler to include the video display and the Tarbell cassette interface, which was very popular back then, going 1500 baud.

"Consequently, the very first standard for public domain programming that I became aware of was based on my personal hardware configuration: the assembler/editor from Processor Technology, the Tarbell cassette, and the Proc Tech video display board. I called it CCOS (Cache Cassette Operating System), made some major changes to it in terms of the editor, and added a *RENUM* command for the assembler program. So we had a little library for cassette-based programs—games, life simulations, etc.," Christensen says.

Much later, in January 1977, Christensen acquired CP/M and worked with his friend, Robert Swartz, who had originally convinced him to buy CP/M and taught him how to use CP/M's built-in assembler and editor. They wrote a little program that would essentially send an entire CP/M diskette, sector by sector, out to a cassette in modem format, meaning in Bell 103 tone. "I think I put a little checksum after each block so that when I got it home, I'd be able to verify that everything had been read OK," he remembers.

He then took that cassette back home and loaded it onto his disk, promptly discovering that CP/M itself had landed across the end of the tape being flipped so he no longer even had a copy of CP/M itself. So, having invented his own disassembler, he took apart some of the other things on the CP/M tape and was able to reconstruct a working CP/M with a program called MOVCPM.COM. He relocated CP/M up to 16K or 20K and got it working again.

Now Christensen finally had CP/M but was still running it on nonstandard floppy disks. By the time 1978 rolled around, he had invented a multifile, load-and-save program for the Tarbell cassette and was distributing public domain programs for it. This cassette-based system was an early way of exchanging files and was exclusive to a local club in the Chicago area called CACHE.

"But I soon realized what a hassle it was doing copies and mailing them, and it just dawned on me that all the pieces again were in place to write a trivial protocol to beep a file over a modem to somebody else," he says. "About this time, the Hayes modem had come out, and it seemed that it was time to try doing something like that. So I just threw the protocol together in a short time!"

"When I started a block, for example, it had to have a start-of-header, a block number, and a block number complement. From a programming standpoint, I wasn't designing the big picture of it. I was saying, 'Let's see, how can I be sure to put in the right block number. Oh, I know, I'll put in a complemented block number rather than doing a CRC on the whole thing.'

"Once I knew that the block number was right, I went to the data and I said I should do some checking on the data. I wonder what people do, maybe just a checksum, I'll just add them up and see, which I regret. Had I done something as trivial as add the carry-in every time I added I would have had an even better reliability.

"But I wasn't out to invent something famous, just to hack something together and get so I could transfer some files."

The original MODEM.COM program was not a very large piece of code. Christensen only implemented three options on it: a terminal mode, an echo mode, and a send-and-receive mode. He then put his new program into the public domain through the CP/M Users Group, and people from all over the country began modifying it.

Mark Zeiger from Flushing, N.Y., added some multifile capability. John Mahr in Chicago added a CRC, so the final version of MODEM7.COM became much more user friendly, menu-oriented, and full-screen-oriented—with CRC and multifile transfer.

But Christensen had long since gotten out of the business of modifying his own MODEM.COM program by then. "I basically kind of threw it into the public domain and then walked away," says Christensen. "I was not the type to devote the rest of my life to supporting every possible modem and everybody's idea of what the program ought to be. I just left it up to them."

"The problem with MODEM.COM early on was that I couldn't just send the program over the phone lines directly to the first people or even give them the disk," he recalls. "I could give a cassette of it to the few people who had my cassette load-and-save program that I put in the public domain earlier."

"Quite frankly, the CP/M Users Group was kind of a one-way street. I would send things into it by cassette, but not many people had compatible disks with my system at the time to send me back anything," says Christensen. "But that was the price of wanting to be the first kid on the block with disks, and I think I was. I'm pretty sure I had mine going well before MITS had shipped the disks for their Altair."

On his original MODEM.COM program and for a few versions thereafter, Christensen was involved in the program's progress within the public domain. He recently discovered that the earliest written date he could find on his program was an assembly language comment dated Sept. 23, 1977.

But after the MODEM4.COM version, Christensen gave up trying to keep up with the program's progress. Yet, after some prodding from a few friends, Christensen finally did break down and give the new MODEM7.COM a try.

"So I got it, and the first time I tried to use it I typed *MODEM7 SODV.600 FILENAME.TYP* and it said 'Invalid Option.' I suddenly realized that my own program didn't even understand the options I programmed in it! And the next thing I typed was *ERA B:MODEM7.COM*!"

Soon to follow the invention of MODEM.COM was another crucial invention created by Dave Jaffe, also from the Chicago area. Called BYE.COM, this new program ran up in high memory and allowed the modem to act as the remote console to a system, permitting people to run commands from a modem on another system.

Christensen got BYE.COM from Jaffe and modified it for the PMMI modem, which was then vastly overtaking Hayes in popularity because it could go 600 baud. He modified it to PMMIBYE.COM and put it in the CP/M Users Group.

During this time, Keith Petersen from Royal Oak, Mich., was often calling into Christensen's CP/M system.

"Keith was the guy who thought user friendliness was important," says Christensen. "And therefore he said he would take MODEM—which was a program that was really meant (particularly with the PMMI) to set the baud rate, set originate and answer mode, prompt all along the way for things like 'Waiting for Sector,' strip it down to its bare minimum components and have only two commands (*Send* and *Receive*), and not change originate/answer or the baud rate."

XMODEM was then born at the hands of Keith Petersen, who took Christensen's MODEM.COM program, stripped it down so that it simply was able to run on a system running BYE.COM using a command as simple as *XMODEM S FILENAME*.

"So I never really invented XMODEM," says Christensen. "The reason XMODEM became equated with my protocol was that people weren't sure whether MODEM was the name of a program or the piece of hardware by the same name! It was just too obscure of a word, and I regret having not thought of something more clever for the original name of it."

"Yet, people would say to me, 'I have a question on your XMODEM protocol,' and I would become staunchly defensive and say, 'Oh, you mean my MODEM protocol?' Eventually I succumbed and I now realize that it's got a new name. And well it should for being able to distinguish from the hardware that's involved."

XMODEM for the IBM PC is an entirely different story for Christensen. After he purchased his first IBM in January 1984, he was obviously interested in picking up a modem program for it, and he found one in the public domain called MODEM7PC.EXE.

"After I got it and tried it out, I realized it didn't work at all at 9600 baud. And it didn't work for the most fundamental of reasons—namely, because the IBM PC screen is so god-awful slow at scrolling that it would say 'Waiting for Sector' often. But by the time the PC had finished scrolling, the header was already past the block that was coming in. So if you cleared the screen and were transmitting a file less than 20 blocks, it would be received," he says.

Christensen now uses a program called Professional YAM (which stands for Yet Another Modem), which was written in C originally for CP/M. It was later significantly enhanced by its author Chuck Forsberg and moved over to the PC as a commercial product.

"Anyway, I admit it, I do get a kick out of seeing my name in books and programs. Every time I see a book on IBM PC communications, for example, I open it up and look for my name," he jokes. "And when I find references to XMODEM, I think 'Oh well, I didn't quite

C

SOFTWARE DEVELOPERS!

V - FILE
THE VIRTUAL MEMORY FILE MANAGER

Let V-FILE save precious development time & cost as you create efficient applications with the power of VIRTUAL MEMORY.

DON'T RE-INVENT THE WHEEL

Why spend weeks or months coding and debugging file and memory management systems when you can order V-FILE today. V-FILE is a library that you can link with your code to provide sophisticated virtual file and memory management — allowing you to concentrate on developing your application.

VIRTUAL DATA OBJECTS SUPPORTED!

Data is referenced by using VIRTUAL MEMORY DATA HANDLES. Your code doesn't need to know whether the data is actually on disk or in RAM. Swapping between disk and RAM and updating files on disk is handled automatically and transparently! Complex VIRTUAL DATA STRUCTURES can be created by linking with data handles instead of pointers.

CHECK THESE FEATURES!

- Multiple, independent swap buffers
- Multiple files per swap buffer
- Highly efficient swap algorithm
- Automatic file updating
- Data prefetching supported
- Data may be locked in memory
- Memory buffers may be flushed
- Makes full use of extended memory on IBM PC/AT
- SOURCE CODE AVAILABLE
- NO ROYALTIES REQUIRED

Supports Dos 2.00+ with
Lattice & Microsoft C compilers
Supports Microsoft windows

$299

V·FILE TM

Contact:
MindBank, Inc.
4620 Henry Street
Pittsburgh, PA 15213
412/683-9800

VISA/MASTER CARD ACCEPTED

CIRCLE 63 ON READER SERVICE CARD

make it in that one.' But then next I'll find a book that has an appendix called The Christensen Protocol and I feel much better!''

Christensen's name is now used as the title for a protocol that's become a standard in the industry. As a by-product of this notoriety, he has had numerous job offers from companies like Lifeboat, NorthStar, and MicroStuff. He has never accepted any, preferring to continue working for IBM.

Because Christensen chose to put his MODEM.COM program into the public domain, he thereby released any legal or commercial control over the program's future. Part of his reason for doing this, he says, was because of his employment at IBM. ''If I were just a programmer at some company, I don't think I'd have conflict-of-interest thoughts running through my mind,'' he says. ''But because I work for IBM and as a result have kind of sold my soul to the company store, I thought I'd be better off giving it away.''

''The protocol is actually not that big a deal,'' he says. ''I mean, it was the right place at the right time. It wasn't that good or anything, it was just the first one.''

Christensen chose to design the protocol around an 8-bit model and went against the attempts by a group called PC-NET, which was trying to establish a 7-bit protocol. ''They were trying to be all things to all people in terms of having a 7-bit protocol capable of efficiently transferring 8-bit data. I think what it amounted to was every 3 bytes having 2 characters or something like that and some encoding with multiplying and shifting, etc.''

''Since CP/M was the software bus, it was a very standard, known environment. You knew what a file looked like—it had 128-byte sectors; it might have had tabs in it if it was ASCII; if it was ASCII it ended with a Control-Z, except for the rare case when it ended right on a sector boundary. You knew what you had and there wasn't anything too inconsistent between that and CP/M systems,'' says Christensen.

''Way back then, I was talking with Don Brown of Potomac Micro Magic (PMMI), who said that 128 bytes was not an unreasonably sized block because if you went with a very short block (say 10 bytes), your overhead would be significant and your turnaround time too, and so on. And if you went with 1K, in the earlier days your line noise and modem quality might cause significant retries.''

But the programming world has changed in certain ways since the mid-1970s, claims Christensen.

''Today, instead of inventing creative solutions to programming problems, you end up spending most of your time seeing if someone else out there has already solved the particular problem before. And after you've found 10 people out there who have done it, you ask yourself if their program will meet your particular needs.

''Admittedly, I think there are still a few niches I can fill. For example, I think there's a need for a good interactive disassembler like my RESOURCE under CP/M. Most disassemblers nowadays seem to be oriented toward an I/O redirected output from DEBUG, which isn't interactive and so on.''

''I'm on my fourth or fifth micro right now,'' says Christensen. ''My Altair gave way to my IMSAI, and I think I went through two Vectors and a little HX-20 along the way. One of the Vectors is still used as my bulletin board. And the IMSAI is still my main CP/M system for maintaining and putting up new versions of CBBS.''

1985 will also be the year of The Hobbyist 2400 he claims. ''With modems now appearing on the market that are comparable in price with 1200 baud modems, I think that people will soon be saying 'why shouldn't we go twice as fast?'''

Christensen looks forward to the future of the digital telephone where telecommunications at 9600 will become routine. He also envisions artificial intelligence as the real long-term future of computing in general.

CP/M-80 C Programmers . . .

Save time

. . . with the BDS C Compiler. Compile, link and execute *faster* than you ever thought possible!

If you're a C language programmer whose patience is wearing thin, who wants to spend your valuable time *programming* instead of twiddling your thumbs waiting for slow compilers, who just wants to work *fast,* then it's time you programmed with the BDS C Compiler.

BDS C is designed for CP/M-80 and provides users with quick, clean software development with emphasis on systems programming.

BDS C features include:

- Ultra-fast compilation, linkage and execution that produce directly executable 8080/Z80 CP/M command files.
- A comprehensive debugger that traces program execution and interactively displays both local and external variables by name and proper type.
- Dynamic overlays that allow for run-time segmentation of programs too large to fit into memory.
- A 120-function library written in both C and assembly language with full source code.

Plus . . .

- A thorough, easy-to-read, 181-page user's manual complete with tutorials, hints, error messages and an easy-to-use index — it's the perfect manual for the beginner *and* the seasoned professional.
- An attractive selection of sample programs, including MODEM-compatible telecommunications, CP/M system utilities, games and more.
- A nationwide BDS C User's Group ($10 membership fee — application included with package) that offers a newsletter, BDS C updates and access to public domain C utilities.

Reviewers everywhere have praised BDS C for its elegant operation and optimal use of CP/M resources. Above all, BDS C has been hailed for it's remarkable *speed.*

BYTE Magazine placed BDS C ahead of all other 8080/Z80 C compilers tested for fastest object-code execution with all available speed-up options in use. In addition, BDS C's speed of compilation was almost *twice* as fast as its closet competitor (benchmark for this test was the Sieve of Eratosthenes).

"I recommend both the language and the implementation by BDS very highly."
Tim Pugh, Jr.
in *Infoworld*

"Performance: *Excellent.*
Documentation: *Excellent.*
East of Use: *Excellent.*"
InfoWorld
Software Report Card

". . . a superior buy . . ."
Van Court Hare
in *Lifelines/The Software Magazine*

Don't waste another minute on a slow language processor. Order your BDS C Compiler today!

Complete Package (two 8" SSSD disks, 181-page manual): **$150**
Free shipping on prepaid orders inside USA.
VISA/MC, COD's, rush orders accepted.
Call for information on other disk formats.

BDS C is designed for use with CP/M-80 operating systems, version 2.2. or higher. It is not currently available for CP/M-86 or MS-DOS.

BDSoftware

BD Software, Inc.
P.O. Box 2368
Cambridge, MA 02238
(617) 576-3828

CIRCLE 4 ON READER SERVICE CARD

Slash Programming Time in Half!

With FirsTime™

- Fast program entry through single keystroke statement generators.
- Fast editing through syntax oriented cursor movements.
- Dramatically reduced debugging time through immediate syntax checking.
- Fast development through unique programmer oriented features.
- Automatic program formatter.

FirsTime is a true syntax directed editor.

FirsTime ensures the integrity of your programs by performing all editing tasks like moves, inserts and deletes along the syntactic elements of a program. For example, when you move an IF statement, FirsTime will move the corresponding THEN and ELSE clauses with it.

Even FirsTime's cursor movements are by syntax elements instead of characters. The cursor automatically skips over blank spaces and required keywords and goes directly to the next editable position.

FirsTime is a Syntax Checker

FirsTime checks the syntax of your program statements, and also:

- Semantics like undefined variables and mismatched statement types.
- The contents of include files and macro expansions.
- Statements for errors as they are entered and warns you immediately.

FirsTime is a Program Formatter

FirsTime automatically indents statements as they are entered, saving you from having to track indentation levels and count spaces.

FirsTime has Unique Features

No other editor offer these features:

The *Zoom command* gives you a top down view of your program logic.

The *View command* displays the contents of include files and macro expansions. This is valuable to sophisticated programmers writing complex code or to those updating unfamiliar programs.

FirsTime's *Transform command* lets you change a statement to another similar one with just two keystrokes. For example, you can instantly transform a FOR statement into a WHILE statement.

The *Move at Same Level command* moves the cursor up or down to the next statement at the same indentation level. This is very useful. For example, you can use it to locate the ELSE clause that corresponds to a given THEN clause or to traverse a program one procedure at a time.

FirsTime is Unparalleled

FirsTime is the most advanced syntax directed editor available. It makes programming faster, easier and more fun.

TO ORDER CALL (201) 741-8188

or write:

Spruce Technology Corporation

189 E. Bergen Place
Red Bank, NJ 07701

In Germany, Austria and Switzerland contact:

Markt & Technik Software Verlag
Munchen, W. Germany
(089) 4613-0

FirsTime for Turbo Pascal $74.95

FirsTime for MicroSoft & IBM Pascal $245

FirsTime for C $295

FirsTime is a trademark of Spruce Technology Corporation • MS-DOS is a trademark of Microsoft Corporation
BM is a trademark of International Business Machines, Inc. • Turbo Pascal is a trademark of Borland International

CIRCLE 26 ON READER SERVICE CARD

"I foresee the day when you will tell your computer to review your in-basket and calendar—it will then tell you the most urgent things you should be doing," he says. "People will probably become quite dependent on these kinds of systems because they'll be an enormous productivity tool."

Christensen also hopes the future will hold a greater opportunity for what he calls "customized deliverables."

"Televisions and things like that are mass marketed, but there may spring up information and entertainment services that are specialized to every possible area of social and intellectual activity," he says. "Somehow, I think there needs to be a way of making things more interesting and more selective to the individual's taste. Maybe that can be done now on such things as CompuServe, where you can do such things as key word searches on articles, etc."

"More important than anything, I have become someone who has more friends who I've never met because of the electronic mediums, because of CBBS, CompuServe, etc. And I think that electronic mail is a very important and useful way for things to happen in the future."

"The problem today with services like CompuServe is that you don't just automatically get your mail; you have to go and get it. The beauty of the U.S. mail is that it's just there when you want it. You can do all this now with a bulletin board, but it's all within a very elite community," says Christensen. "I think a big problem also is that bulletin boards are basically single-user, until you get into the UNIX ones that are multiuser, and as a result they're often busy and therefore cannot be considered a viable utility."

Christensen also says he thinks the United States Postal Service could well become the medium for a nationwide electronic network. "Just in the fact that they have so many physical buildings, they could start putting computers in them. And, as the volume of paper mail goes down and the volume of electronic transfers go up, they would then be nicely situated to make a slow conversion to electronic types of mail."

On the subject of bulletin board and network security, Christensen is an adamant supporter of keeping systems publicly open as much as possible. "A lot of them are just going private today, and I'm trying like crazy to keep the world's oldest BBS open on your first call so you can leave a message and do what you like," he says. "I do that by having both myself and a dozen other people who call on a very regular basis to police the system. But it's enormously frustrating to have such blatantly obscene and inappropriate messages put up as often as they are."

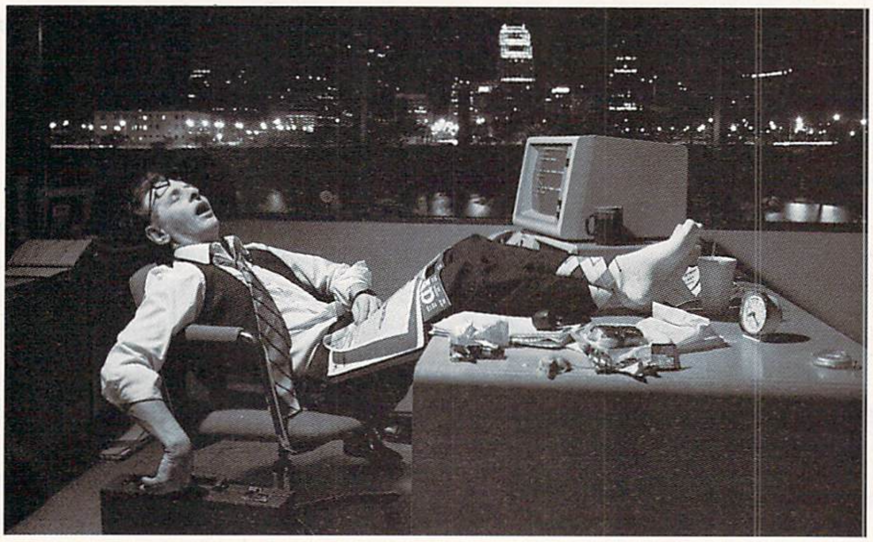
"It almost makes me wonder if there haven't already been some electronic banking rip-offs that have just not been publicized because it would shake up too many people," says Christensen. "Like, for example, what just happened in Ohio where one state-chartered savings and loan closed and its closing put a run in people's minds. I'm not saying there'd ever be such a run, but the banking system would always be trying to protect its image against such situations ever becoming known."

Christensen is also a strong opponent of copy-protected software. "I find it just significantly inconvenient," he says. "Since the world was able to get along with it under CP/M, I think I could get along without it now. Quite frankly, I refuse to trade some company's paranoia against my inconvenience."

"I guess basically it comes down to something even more fundamental than any of that," he says. "I consider myself a very high-integrity, honest person and resent being reminded of how corrupt the world is." ■

Craig LaGrow is Editor of COMPUTER LANGUAGE.

If you're tired of waiting, you're using the wrong file manager.



Be fast. Btrieve.™

If batch jobs and reports turn waiting time to nighttime, then wake up! You're using the wrong file manager.

Btrieve™ file management hates waiting as much as you do. It's written in assembly language especially for the IBM PC. And based on b-tree file indexing, with automatic balancing for access speed that won't degrade as your database grows. With Btrieve, your applications always run fast. So you'll be out the door faster.

The standard for networking. Btrieve/N (network version) sets the standard for the industry's most popular LANs, including IBM's PC Network. Btrieve/N offers *safe* network file management that coordinates simultaneous updates and prevents lost data.

Automatic file recovery. Btrieve provides automatic file recovery after a system crash. Your Btrieve data always comes back intact.

Fully-relational data management. SoftCraft's entire family of products gives you a complete, fully-relational database management system. Rtrieve™ adds report writing capabilities. Xtrieve™ speeds users through database queries with interactive menus.

For professional programmers. Btrieve is the fast, reliable answer for all your application development in BASIC, Pascal, COBOL, C, FORTRAN, and APL. With Btrieve, you can develop better applications faster. And know they'll run—*fast*.

SoftCraft Inc.
P. O. Box #917 Austin, Texas 78766
(512) 346-8380 Telex 358 200

Suggested retail prices: Btrieve, $245; Btrieve/N, $595; Xtrieve, $195; Xtrieve/N, $395; Rtrieve, $85; Rtrieve/N, $175. Requires PC-DOS or MS-DOS 1.X, 2.X, or 3.X.

CIRCLE 38 ON READER SERVICE CARD

THE fifth generation language

P R O L O G

Implementing the full Edinburgh Syntax as described by Clockson and Mellish. Recognized by Japan as providing unparalleled opportunity for artificial intelligence.

Applications:

- ▲The highest level of a heirarchial robotic control system.
- ▲Machine recognition of natural language.
- ▲Expert systems and knowledge engineering.

Optional:

- ▲Special libraries
- ▲Language extensions

Educational Package

$29.95

(MSDOS)

Highlights of LVM Prolog:

- ▲Type LVM Prolog makes possible the execution of AI applications previously only possible on a mainframe.
- ▲Invisible compilation to a semantic network provides the flexibility of the interpreted mode and the speed of a compiler.
- ▲Virtual memory and a sophisticated cache algorithim limited only by the DOS, typically 2 gigabytes.
- ▲Syntax superset with many extensions such as pattern specified insertion and deletion, clause indexing, robotic control.
- ▲Unlimited number of resident and virtual modules, each up to 2 gigabytes in size.

▲ ▲ ▲ ▲ ▲ ▲

▲THE most cost efficient package in the industry.

$500- (MSDOS version) Also available for Xenix, Unix, CP/M68K

automata design assoc.

▲VISA, Mastercard, AMEX

▲call or write for brochure

1570 Arran Way Dresher, PA 19025 **technical:** (215)646-4894 **orders:** (215)355-5400

CIRCLE 1 ON READER SERVICE CARD

the source debugger for lattice C

Your time and convenience come first! The MSD C Debugger™ is the last, and perhaps final, word in programming assistance for Lattice C users. C Debugger produces a high level view of C programs via function names, line numbers, variable names and C data types, plus a low-level view of machine addresses and instructions for testing assembler language functions.

More features include:

- All documentation is prepared for programmers.
- Online help screen throughout the process.
- Capability to single step through your program.
- Set break points, examine registers and variables.

$165.00 **+$3.50 shipping**

VISA MasterCard

MSD

To order, call or write:
MICRO-SOFTWARE DEVELOPERS, INC.
214½ W. Main St. • St. Charles, IL 60174
312/377-5151

Lattice C is a trademark of Lattice, Inc.

CIRCLE 54 ON READER SERVICE CARD

Now With Windowing!
$49.95 Basic Compiler

MTBASIC

Features:

Multitasking	Windowing
Handles interrupts	Interactive
Fast native code	Compiles quickly
Floating point	No runtime fee

MTBASIC is a true native code compiler. It runs *Byte's* Sept. '81 sieve in 26 seconds; interpreters take over 1400 seconds! Because MTBASIC is multitasking, it can run up to 10 Basic routines at the same time, while displaying ten separate windows. Pop-up/down menus are a snap to implement.

The MTBASIC package includes all the necessary software to run in interpreter or compiler mode, an installation program (so *any* system can use windowing), three demonstration programs and a comprehensive manual.

AVAILABLE for CP/M (Z-80), MS-DOS, and PC-DOS systems.

ORDERING: Specify format when ordering. We accept Visa, MC, checks and COD. Send $49.95 plus $3.50 shipping and handling ($10 overseas) to:

SOFTAID, Inc.

P.O. Box 2412 Columbia, MD 21045-1412
301/792-8096

CIRCLE 78 ON READER SERVICE CARD

ORDER COMPUTER LANGUAGE BACK ISSUES WHILE THEY LAST!

Only a limited quantity of magazines is available, so order today. To receive your back issues, just fill out this coupon and mail it back with a check for $4.00 per issue.

Premier ____ copies × $4.00 = $ ______
Oct. '84 ____ copies × $4.00 = $ ______
Nov. '84 ____ copies × $4.00 = $ ______
Dec. '84 ____ copies × $4.00 = $ ______
Jan. '85 ____ copies × $4.00 = $ ______
Feb. '85 ____ copies × $4.00 = $ ______
Mar. '85 ____ copies × $4.00 = $ ______
Apr. '85 ____ copies × $4.00 = $ ______
May '85 ____ copies × $4.00 = $ ______
Jun. '85 ____ copies × $4.00 = $ ______
Total = $ ______

Foreign orders: Add $3.00 for airmail.

NAME ______________________
COMPANY ______________________
ADDRESS ______________________
CITY, STATE, ZIP ______________________

6

Send payment and coupon to:
Computer Language
Back Issues
131 Townsend St.
San Francisco, CA 94107

pluto T.M.

PLUTO—MORE THAN JUST A BASIC!

PLUTO is a powerful, multiple user Business Basic programming environment for the IBM PC, XT, AT and the PC compatibles.

In addition, PLUTO interprets MAI/Basic Four and Science Management Corporation SMC Business Basic programs to run on microcomputers.

Powerful features of PLUTO not found in other Basics:

- Multiple user capabilities on DOS 2.0 (and above)
- Indexed and direct files
- Access to host system text (serial) files
- Record and file locking
- Trigonometric, logarithmic and exponential functions
- Menu driven resource configurator
- Multidimensional arrays and vectors
- Binary and string logical functions
- Extended position function
- Public programming
- Supports ghost tasks
- Fully formatted output
- Up to 255 files per program

All of this for only **$595.00.** CALL TODAY!

SOUTHWEST DATA SYSTEMS, INC. 3017 San Fernando Blvd., Burbank, CA 91504 • (818) 841-1610

CIRCLE 46 ON READER SERVICE CARD

SPARRY BASIC-B COMPILER

1. Floating Point Math
2. Use all 640K of Memory
3. Multiple Data Segments
4. Multiple Code Segments
5. Internal ISAM File support
6. 4 Virtual Screens (Big Windows)
7. Easy Assembly Language Interface
8. Direct System Interrupt Calls
9. A Compatible BASIC Compiler

Req. PCDOS 2.00 + with 128K

Sparry Software Labs

P.O. BOX 632
MILFORD, MA 01757
617-473-5435

( ) Compiler $159
( ) Demo Disk $15

PCDOS is a Trademark of
International Business Machine Corp.

CIRCLE 66 ON READER SERVICE CARD

DEZIGN

THE STRUCTURED PROGRAMMING TOOL FOR MODERN TIMES

- Design your programs right on the screen, using modern techniques based on the popular Jackson Structured Programming method (JSP)!
- **DEZIGN** is more than just another flowcharting tool. It is an integrated tool for designing and documenting programs and for generating ADA, C, PASCAL, and PL/I source code, as well as dBASE II and dBASE III command files.
- **DEZIGN** enables you to create Data and Program Structure Diagrams using the Sequence, Selection (IF-THEN-ELSE), and Iteration (DO WHILE) constructs; assign detailed statements to the diagrams; and synthesize source code from the control logic represented on the diagrams and the detailed statements assigned to them.
- **DEZIGN** lists for $200. It runs on the IBM PC, XT, or AT and requires 128K RAM, one disk drive, and an 80-column color or monochrome display.
    - DEZIGN-PC runs under DOS 2.0, 2.1, and 3.0.
    - DEZIGN-86 runs under CP/M-86 1.1.
- Want to learn more? Please contact us concerning pricing and availability of JSP reference texts and seminars.

ZEDUCOMP • P.O. BOX 68 • STIRLING, NJ 07980

(201) 755-2262

dBASE II and dBASE III are trademarks of Ashton-Tate, Inc.

CIRCLE 83 ON READER SERVICE CARD

WRITE

The Writer's Really Incredible Text Editor lives up to its name! It's designed for creative and report writing and carefully protects your text. Includes many features missing from WordStar, such as sorted directory listings, fast scrolling, and trial printing to the screen. All editing commands are single-letter and easily changed. Detailed manual included. Dealer inquiries invited. WRITE is $239.00.

BDS's C Compiler

This is the compiler you need for learning the C language and for writing utilities and programs of all sizes and complexities. We offer version 1.5a, which comes with a symbolic debugger and example programs. Our price is (postpaid) $130.00.

Tandon Spare Parts Kits

One door latch included, only $32.50.
With two door latches $37.50.
Door latches sold separately for $7.00.

All US orders are postpaid. We ship from stock on many formats, including: 8", Apple, Osborne, KayPro, Otrona, Epson, Morrow, Lobo, Zenith, Xerox. Please request our new catalog. We welcome COD orders.

Workman & Associates

112 Marion Avenue
Pasadena, CA 91106
(818) 796-4401

WORKMAN & ASSOCIATES USER-FRIENDLY SOFTWARE

CIRCLE 68 ON READER SERVICE CARD

PRODUCT BINGO

By Doug Millison

Each month Product Bingo features the latest in new software and hardware products of interest to COMPUTER LANGUAGE readers. Product Bingo items are based on information received from the manufacturer and are not meant to be product evaluations, reviews or endorsements. To find out more about a particular product simply circle the appropriate number on the Reader Service card—you'll receive information directly from the manufacturer.

Note to manufacturers: Send new product information to Doug Millison, Product Bingo, COMPUTER LANGUAGE, 131 Townsend St., San Francisco, Calif. 94107.

New tools for C, AI programmers

From **CCA Uniworks** comes the **Safe C Family**, consisting of run-time analyzer, interpreter, dynamic profiler, English-to-C, and C-to-English translators. Also introduced are three new languages for AI programming. **COMMON LISP** is a flexible version of LISP. **OPS5** is a new VAX language. **MPROLOG** is based on PROLOG. Prices were unavailable at press time.

CCA Uniworks, 20 William St., Wellesley, Mass. 02181, (617) 235-2600.

CIRCLE 101 ON READER SERVICE CARD

Software development under UNIX

Software Development Systems Inc. announces significant enhancements to the **UniWare Software Development System** for development under UNIX.

Improvements include an absolute-listing generator, more sophisticated link editor, object module librarian, and others.

Software Development Systems Inc., 3110 Woodcreek Dr., Downers Grove, Ill. 60515, (312) 971-8170.

CIRCLE 102 ON READER SERVICE CARD

New Microsoft C compiler, Macintosh BASIC

Microsoft Corp. has released a new C compiler, promising increased execution speed and program compactness. Suggested retail price is $395. Version 2.0 of Microsoft BASIC for Macintosh accesses all unique user interface features. Suggested retail price is $150.

Microsoft Corp., 10700 Northrup Way, Box 97200, Bellevue, Wash. 98009, (206) 828-8080.

CIRCLE 103 ON READER SERVICE CARD

New C profiler, compiler

Catalytix Corp. has released the **Safe C Dynamic Profiler** and the **Safe C Compiler.**

Prices for the Safe C Dynamic Profiler range from $200 for MS-DOS to $1,500 for VAX 11/780. Prices for the Safe C Compiler range from $400 for MS-DOS to $4,000 for VAX 11/780.

Catalytix Corp., 55 Wheeler St., Cambridge, Mass. 02138, (617) 497-2160.

CIRCLE 104 ON READER SERVICE CARD

FORTRAN graphics library for IBM PC

Tekmar, a FORTRAN library allowing powerful graphics functions to be called from programs, has been released by **Advanced Systems Consultants.**

The Tekmar graphics package, including FORTRAN object libraries and application source, is $195.

Advanced Systems Consultants, 18653 Ventura Blvd., Ste. 351, Tarzana, Calif. 91356, (818) 990-4942.

CIRCLE 105 ON READER SERVICE CARD

BaZic86 links NorthStar, IBM PC

Micro Mike's Inc. introduces **BaZic86** and several utilities for NorthStar and IBM PC users. BaZic86 allows NorthStar BASIC of BaZic80 programs to run on the IBM PC family and other MS-DOS computers. Also available are utilities allowing transfer of NorthStar BASIC programs and data files from Northstar DOS to CP/M formats.

Micro Mike's Inc., 3015 Plains Blvd., Amarillo, Texas 79102, (806) 372-3633.

CIRCLE 106 ON READER SERVICE CARD

New multiuser, multitasking system

From **Inner Access Corp.** comes the **MultiUser-16** computer system. MultiUser-16 supports from eight to 64 users, with .5MB memory expansible to 16MB.

MultiUser-16 is priced at $8,995 to authorized dealers and VARs.

Inner Access Corp., 517K Marine View, Belmont, Calif. 94002, (415) 591-8295.

CIRCLE 107 ON READER SERVICE CARD

Extended PIP utility released

XPIP, a system utility released by **System Facilities Inc.**, provides an easy-to-use interface for users of MS-DOS, PC-DOS, and CP/M operating systems.

XPIP is priced at $29.95.

System Facilities Inc., P.O. Box 7079, Charlottesville, Va. 22906, (804) 977-5245.

CIRCLE 109 ON READER SERVICE CARD

SuperSoft Diagnostics

When Reliability Counts

Protect yourself from time-robbing system failure. Pinpoint costly hardware problems before they cause serious trouble. Diagnostics II from SuperSoft can help you eliminate hardware problems, service calls, and data loss due to system failure.

End Users

Diagnostics II is the finest set of system diagnostics available for microcomputers. It thoroughly checks memory, CPU, terminal, printer, and disk drives – isolating many problems to the chip level. It checks both standard and non-standard components, including non-IBM add-ons. The memory test is particularly powerful; incorporating a quick test, walking bit test, burn-in test, and speed test to make sure every bit of memory is completely reliable.

Manufacturers

Hardware manufacturers, systems houses, and service organizations – we can tailor our diagnostics software to your specific needs. We have developed custom diagnostics for companies such as NCR, XEROX, MORROW DESIGNS, and SONY. From easy to operate user level diagnostics to exhaustive service level tests, we can provide the expertise you need.

So whether you're an end user, service technician, or system manufacturer, get SuperSoft's Diagnostics II for yourself and keep your system in great shape.

Diagnostics II
(for all PC DOS, MS DOS, CP/M-86, and CP/M-80 systems): $125
Call for pricing on customized versions.

TO ORDER CALL
800-762-6629
(in Illinois call 217-359-2112)

or SEND YOUR CHECK OR CREDIT CARD INFORMATION TO THE ADDRESS BELOW. Add $3 shipping U.S., $6 Canada, $20 all other areas. Please specify your computer and operating system. (C.O.D. orders also accepted)

SuperSoft

SuperSoft, Inc. P.O. Box 1628,
Champaign, IL 61820
Telex: 270365 SUP ACI CHM

CIRCLE 77 ON READER SERVICE CARD

Write it once!

MasterFORTH

Portable programming environment

Whether you program on the **Macintosh**, the **IBM PC**, an **Apple II** series, a **CP/M** system, or the **Commodore 64**, your program will run unchanged on all the rest. If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

MasterFORTH is a state-of-the-art implementation of the Forth computer language. Forth is interactive – you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in macro assembler to make it even faster. MasterFORTH's relocatable utilities, transient definitions, and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint, and trace your way through most programming problems. A string package, file interface, and full screen editor are all standard features.

MasterFORTH exactly matches the Forth-83 Standard dialect described in *Mastering Forth* by Anderson and Tracy (Brady, 1984). The standard package includes the book and over 100 pages of supplementary documentation.

MasterFORTH standard package	
Macintosh	$125
IBM PC and PC Jr. (MS DOS 2.1)	125
Apple II, II+, IIe, IIc (DOS 3.3)	100
CP/M 2.X (in several formats)	100
Commodore 64	100
Extensions	
Floating Point (1984 FVG standard)	$40
Graphics (Apple II series)	40
Module relocator (with utility sources)	60
Printed source listing (each)	35
Publications	
Mastering Forth (additional copies)	$18
Thinking Forth by Leo Brodie	16
Forth-83 International Standard	15
Rochester Bibliography, 2nd ed.	15
1984 Rochester Conference	25
1984 Jl of Forth Appl. & Res. 2(2)	15
1983 FORML Conference	25

MICROMOTION

12077 Wilshire Blvd., #506
Los Angeles, CA 90025
(213) 821-4340

CIRCLE 64 ON READER SERVICE CARD

ConIX

Hardware Requirements: CP/M 2.2 or equivalent system (including emulators)
Price: $165
Available from: Computer Helper Industries Inc., P.O. Box 680, Parkchester Station, N.Y. 10462, (212) 652-1786.
Support: Telephone and mail support included in price, updates $20

There has been a great deal (this may be the understatement of the century) of coverage in the last few years on the UNIX operating system. Most of this has been laudatory, with the result that nearly everyone now coming out with new machines is putting some form of UNIX work-alike out with it. Those of us with older machines, running with CP/M on a Z80, seem to be left out in the cold. We suffer a lack of more modern features that the UNIX people have, in exchange for maintaining compatibility with our already extant large software base.

Naturally enough, a number of software entrepreneurs found the situation intolerable. They asked themselves if there was any way to graft onto CP/M the desirable features of UNIX, such as I/O independence and a programmable command interpreter with I/O redirection, while still not interfering significantly with existing programs. Not surprisingly, the answer to this question turned out to be a resounding "yes!"

The market soon became flooded with replacement command interpreters for CP/M which supported most, if not all, of the better features of the UNIX shell. Having worked with quite a few of these products (and written one myself), I can say that all are successful to some degree, but most have flaws. When Console Input eXecutive (ConIX) was made available to me for review, I did not expect to be surprised, but I was and, on the whole, pleasantly so.

ConIX by Computer Helper Industries Inc. in New York, N.Y., is not just another command control processor (CCP) replacement. It supports all the standard CP/M CCP features but goes a great deal farther in flexibility than any of the other CP/M shells I've seen.

It goes beyond them by extending the BDOS itself to provide extra CP/M function calls for accessing the ConIX data areas and functions. It supports the usual features of console I/O redirection and shell programming language, but even these have some twists that were pleasant surprises.

The package came to me with a standard 8-in., single-sided, single-density CP/M disk, a very nice manual and a licensing agreement/warranty. The agreement was one of the more reasonable of its genre I've seen, although still drenched in legalese, a language that could never be programmed.

The manual is exceptionally well-designed, with a complete table of contents and index, and is printed in a standard typeface. I'm one of those people who read the manual first, shocking as that may be to some, and I found it very clear and concise in most areas.

It took me a while to get through it; it's about an inch thick. When I finished I found that I still had nine pages of extra information to print out of the .DOC files on the disk. Most of the addenda consisted of still more features that had been added after the manual was printed.

When I put the manual down, the phrase "rampant optionitis" kept running through my mind. If all you count is number of options and built-in commands, this system has all of its competitors beat. Of course, just because it has all these options does not mean that you must use them all or even need to remember them. The options are there if you need them, and the very good table of contents and index makes finding them very easy.

Installation is a snap. The manual and a .DOC file provide a step-by-step process. At first I simply took the defaults. The system is provided in a format much like DDT or ZSID, and one of the things the install process does is generate a CONIX.COM file sized to your specifications so that relocation is not necessary each time the system is loaded.

The install process also allows you to disable the two features of ConIX that might cause problems. ConIX will patch the BIOS vectors in order to do I/O redirection in certain cases. Some systems cannot tolerate this, so one INSTALL option is to disable BIOS patching.

I had a problem here. Both my CCP and BIOS were nonstandard, and this caused INSTALL to generate a ConIX that went off into east hyperspace when loaded. When I went back to my vanilla CP/M, as I got it from the people who made my disk controller, everything went as specified.

The other feature that might cause trouble is the ExpanDiskVirtual Floppy option. If this one has to be disabled in your system, I commiserate with you. This option lets you access all 16 logical drives that CP/M allows even if you only have two physical drives.

Any references to the virtual drives causes a mount message to be issued and the logical drive to be logged in and associated with a physical drive. This means the "Disk Full" message loses a lot of its terror since you can mount a fresh disk where the full one was. Very handy and unique to ConIX.

Once installed, ConIX is invoked just as any other program. It comes up with a copyright notice and then looks for a program called PROFILE.COM. This is usually a ConIX command language program, which sets all the default options and does whatever initialization processes the user wants, such as issue a log-on message. (The .COM might be confusing, but I'll get to the unique aspects of ConIX command language programming later.)

When PROFILE terminates, ConIX types its characteristic prompt and waits for the user commands. My only complaint is that the manual does not clearly explain the function of the PROFILE program. I wasted a lot of time trying to figure out why my keyboarded option settings were vanishing under certain conditions.

The utility programs that come with ConIX are also unique to this type of package and prove to be very useful. ARM is a file archiver, similar in function to the famous public domain program LU. Using ARM, the user stores files in a library format; thus, related data can be stored in a single place.

Since CP/M allocates disk space in fixed size blocks, storage of small files in a library can also save a significant amount of disk space. For instance, on my system, the smallest possible unit of disk space is 2K. If I have a file only 100 bytes long, most of that 2K is wasted. If I have 10 files, each 100 bytes long, then the total space I need is only 2K, but CP/M will force me to use 20K of disk space.

Using ARM, I can gather these 10 files into a library that may take only 4K (2K for the actual data and 2K for control information). This is nice enough; additionally, ConIX allows you to specify that libraries be searched for programs automatically. If you have a lot of little utilities lying around, you can put them in a library that ConIX will search when you invoke the program in the usual manner.

The archiver also has time- and date-stamp capability and will store a short description of each member when it is added to the library. I was unable to cause it to malfunction, but I did note that when a large number of files were in the library, performance was rather poor. On a fast floppy or a hard disk, it probably would not be noticeable. However, on my venerable Shugart 801s, it was.

The MAN utility should be familiar to UNIX fans out there. The utility gives on-line help on the ConIX commands. The MANual files are created and maintained by ARM. If you forget an option or command, the command *MAN – <manual file>* gives a table of contents of the manual file, and *MAN <subject> <manual file>* gives a detailed explanation of the subject from the specified manual file.

The MANual system is a necessity in a system this rich. Fortunately, the manual files don't use that much space. ARM is apparently very efficient in its library format.

The printed ConIX manual is very well-organized. However, after a while, I found it was easier and faster to use the on-line help feature. The fact that ARM is used to build the help files makes it possible to easily add help entries for your own programs or modify existing entries if you feel they are a little less clear than they could be.

The built-in commands and features are the real stars of this show, though. These range from the standard *DIR* command to I/O port access from the command line. There are shell variables that can be used to store strings and numbers, and the string variables are nonvolatile, being written to disk.

Shell variables are also used to implement programmable function keys. You can have up to 52 (A-Z, a-z) arbitrary strings stored on disk, ready to recall at the touch of a user-selectable lead-in key.

An example of this might be helpful. My terminal will generate a *0D0H* when a special key is hit, so I decided that this key would be appropriate as a lead-in character. Using the built-in option command, *OPT*, I assigned this value as the PFK lead-in byte: *opt +fk 0d0*. Then I used the *SET* command to assign a string to the shell variable *$A*:

```
set a = "dir b:^m
```

After this, hitting <special>A generates *dir b:* with a <return>. As far as any program, using either BDOS or BIOS console I/O, is concerned, I typed that in.

Other internal commands allow direct manipulation and examination of memory, the ability to load executable files at any address, and saving memory to disk from any address—thus eliminating the need for the CP/M *SAVE* command. They also do general file management, copy files, type files, erase files, etc., without invoking separate programs.

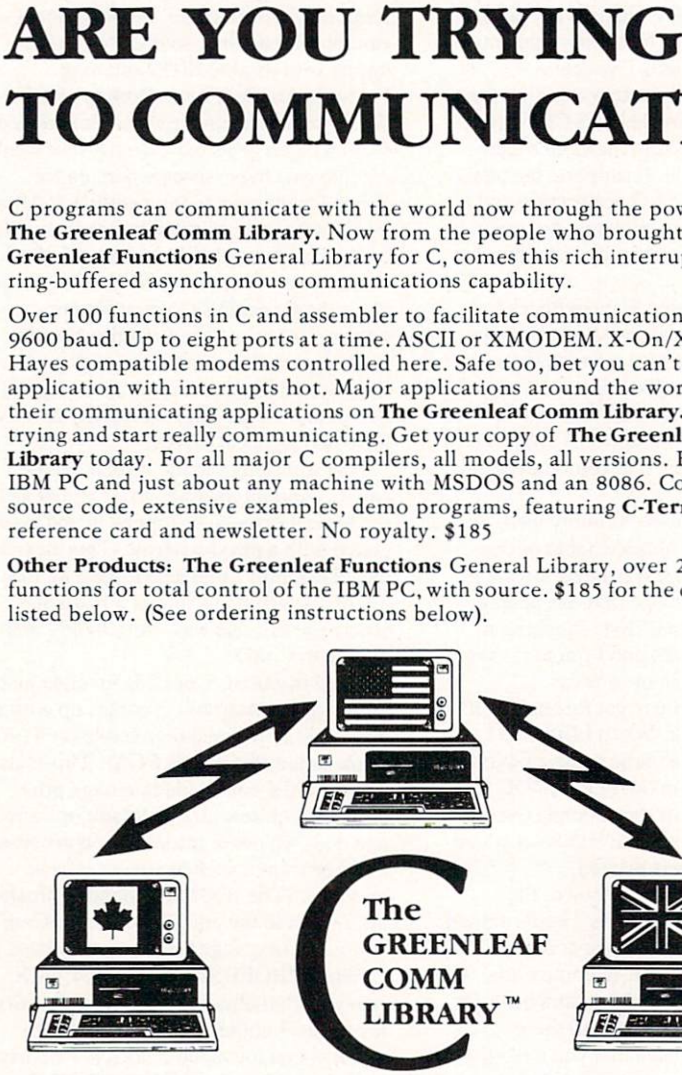
CP/M's various user areas are directly accessible since ConIX extends the standard file name to include a user-area designator. *B:15/ZAP.COM* references the file ZAP.COM on drive B, user-area 15.

ARE YOU TRYING TO COMMUNICATE ?

C programs can communicate with the world now through the power of **The Greenleaf Comm Library.** Now from the people who brought you **The Greenleaf Functions** General Library for C, comes this rich interrupt driven, ring-buffered asynchronous communications capability.

Over 100 functions in C and assembler to facilitate communications at up to 9600 baud. Up to eight ports at a time. ASCII or XMODEM. X-On/X-Off too. Hayes compatible modems controlled here. Safe too, bet you can't exit your application with interrupts hot. Major applications around the world base their communicating applications on **The Greenleaf Comm Library.** Stop just trying and start really communicating. Get your copy of **The Greenleaf Comm Library** today. For all major C compilers, all models, all versions. For the IBM PC and just about any machine with MSDOS and an 8086. Comes with source code, extensive examples, demo programs, featuring **C-Terminal,** reference card and newsletter. No royalty. $185

Other Products: The Greenleaf Functions General Library, over 220 functions for total control of the IBM PC, with source. $185 for the compilers listed below. (See ordering instructions below).



The GREENLEAF COMM LIBRARY™

Specify compiler when ordering: Lattice, Microsoft, Computer Innovations, Mark Williams, or DeSmet. Add $7.00 for UPS Second Day Air (or $5.00 for ground). Texas residents add sales tax. Mastercard, VISA, check, or P.O. In stock, shipped same day.

☐General Libraries	$185
☐Comm Library	$185
☐CI186 Compiler	$349
☐Lattice C	$395
☐Mark Williams	$475

PRICES ARE SUBJECT TO CHANGE WITHOUT NOTICE.

For Information:
214/446-8641

GREENLEAF SOFTWARE©

2101 HICKORY DR.
CARROLLTON, TX 75006

CIRCLE 44 ON READER SERVICE CARD

All the ConIX utilities will accept this form, but most CP/M programs will not. This can be a real problem, but it is possible to get around it by a little creative use of ConIX command files, of which more will come later.

There is an option to extend the number of user areas accessible from the command line from the 16 that CP/M normally allows to 32.

There is also a command to list directories of other user areas, no matter what user area is current. This is something I've felt was missing from CP/M, and it's nice to find that others feel the same way.

Like other products of this type, ConIX supports I/O redirection to or from a program. It goes farther than the others in the way it implements the feature and in the options possible. Console I/O is redirectable in the usual manner; unusual is that printer I/O is also redirectable (incidentally, the package includes a built-in print spooler as well).

ConIX also supports the unique ideas of memory files and memory I/O. This means that if you have the memory space, you can redirect I/O to or from memory buffers, as if the memory were a device. You can specify the addresses and sizes of the buffers and then use them like small files at a rather astonishing performance advantage.

You can even pipeline the output of one program into the input of another using memory, which is more like the UNIX concept of a pipe than is usually possible on CP/M-based machines. Of course, a problem remains; this is still basically CP/M, so one program must finish before the other can start. If the memory buffers overflow, the system can get written over, and the user gets the privilege of hitting the RESET button.

Speaking of pipelines, ConIX has an approach to this very useful feature that I wish I had thought of. Since CP/M is a single-thread operating system, true UNIX-style pipes are impossible, so pipes are implemented as temporary files that are managed by the system:

```
prog1 | prog2
```

is rendered (invisibly) as

```
prog1 > temp.fil; prog2 < temp.fil;
    era temp.fil
```

ConIX is different from other products of its type, in that to maximize performance, you have an option in where the invisible temporary file is placed. If you have a memory drive or a hard disk, placing the temporary pipe file there can yield practically instantaneous throughput.

Most of the other CCP replacements improve on the *SUBMIT* capability of CP/M. Some merely support straight-batch processing, while others have a complete command language with decision capabilities, looping constructs, etc.

ConIX is one of the latter, with a difference. The shell programs are compiled into a pseudocode and then executed interpretively. The output of the compiler is a .COM file with a special header. After compilation, the user cannot tell if the program executing is true machine code or a ConIX shell script. (A compiled shell script will not execute under pure CP/M, however. The first instruction is a *RET*, which simply sends you back to the CCP.)

As with regular .COM files, you have the advantage of archiving the compiled shell scripts into the default library and then executing them in the normal manner. Since the scripts do not have to be kept in text form, you can reclaim a lot of disk space.

In many ways the command language is more a programming language than a classical command language. It has *WHILE* loops, *IF-THEN-ELSE* decisions, *SWITCH* multiway branches, error trapping, keyboard-interrupt processing, subroutines (with *GOSUB <label>* and *RETURN*) and even the ubiquitous *GOTO <label>* statement. The *AND* and *OR* logical operators are implemented also.

One possible problem in the system is that all the conditionals test the ConIX exit status, which the built-in commands and utilities set. Ordinary CP/M programs do not set this status, so the conditionals can't directly be used on the typical user program. Ways around this exist. The one suggested in the manual is to capture output from the program and test for known strings in this output. Though clumsy, it does work.

I wish I had room here to show one of the command programs that came with the package, a menu-driven, general-purpose utility program reminiscent of SWEEP. Unfortunately, the thing takes about five pages and has options for examining and altering files, memory, and I/O ports, copying files, interactively deleting files, copying memory to disk, etc. It's impressive. A somewhat shorter example command language program is in Listing 1. This is a quick and dirty routine I wrote to send a variable number of form feeds to my printer. It illustrates some aspects of the command language.

```
if scmp "$1" \<\> ""; then
   echo "$1\\" > @0040-0041
else
   echo "1\\" > @0040-0041
endif
while - test $@0040 0; do
   echo ^l > :lst
   sub $@0040 1 > @0040-0041
end
```

Listing 1.

REMOVE

LINT

from your **C** programs
with

PC–LINT

PC–LINT analyzes your C programs (one or many modules) and uncovers glitches, bugs, quirks and inconsistencies. It will catch subtle errors before they catch you.

PC–LINT resembles the Lint that runs on the UNIX O.S. but with more features and greater sensitivity to the problems of the 8086 environment.

- Full K&R C
- Supports Multiple Modules—finds inconsistencies between declarations and use of functions and data across a set of modules comprising a program.
- Compares function arguments with the associated parameters and complains if there is a mismatch or too many or too few arguments.
- All warning and information messages may be turned on and off globally or locally (via command line and comments) so that messages can be tailored to your programming style.
- All command line information can be furnished indirectly via file(s) to automate testing.
- Use it to check existing programs, programs about to be exported or imported, as a preliminary to compilation, or *prior* to scaling up to a larger memory model.
- All one pass with an integrated preprocessor so it's very fast.
- Has numerous flags to support a wide variety of C's, memory models, and programming styles.
- **Introductory Price: $98.00 MC, VISA**
  (Includes shipping and handling) PA residents add 6% sales tax. Outside USA add $10.00.
- Runs on the IBM PC (or XT, AT or compatible) under DOS 2.0 and up, with a minimum of 128KB of memory. It will use all the memory available.

GIMPEL SOFTWARE

3207 Hogarth Lane • Collegeville, PA 19426
(215) 584-4261

*Trademarks: IBM *(IBM Corp.)*, PC–LINT *(Gimpel Software)*, UNIX *(AT&T)*

An explication is in order. The program is invoked by *tof <n>*, where *n* is the number of form feeds desired. If *<n>* is missing, it defaults to one. The first line checks for the presence of an argument. Note that *$1* is used in the same manner that *SUBMIT* uses it. The backslashes are escape characters needed to prevent interpretation of < and > as redirection operators.

SCMP and *ECHO* are built-in functions, the former initiating a string comparison and the latter simply printing its arguments. Note the memory redirection, storing the value in the command line at address *$0040*. The *TEST* command compares the value at *$0040* with 0 and sets a status flag, which the – operator then inverts, so this is a *while not zero* loop.

In the loop body, *ECHO* copies a form feed to the standard output, which is redirected to the *:LST* device, then the *SUB* command subtracts one from what is stored at $0040 and stores the result back there. The executable version of the code in Listing 1 is less than 128 bytes long.

All of this power does not come without a price. ConIX takes up more than 28K. This means that unless something can be done to alleviate memory constraints, a great many standard programs simply will not run.

Happily, the ConIX architect (I can't believe that something this clean was thrown together by a committee!) has a solution to this. When an external program is invoked, ConIX frees a user-specifiable amount of memory space for the program's use and, after program termination, will reload the segments of itself that were freed. This memory management is necessary for a program this size and seems to be very well implemented.

Eight levels of management exist, ranging from level one, which, basically, frees nothing but allows a user program to use all aspects of the ConIX internal commands, to level eight, which frees everything except a small system loader. One of the option (*OPT*) commands will set the default level. For any individual program, a command-line option sets the memory-management level for that invocation alone.

After your program executes, the loader will reload the freed segments and restart ConIX. This memory management is invoked only for running .COM-type files that are not compiled shell scripts identified via the special header.

The manual needs work in this area. I found myself spending a great deal of time doing things like shifting default management levels around before it became clear what was going on.

User support is very important in the software game, especially for a package as complex as ConIX. I called Computer Helper Industries twice during the course of the review process, without identifying myself as a reviewer. Both times my questions were answered quickly and courteously, a pleasant change from some other companies I've had to call. The person I spoke with seemed to be genuinely concerned with solving any problems I might have. This is a definite plus in favor of Computer Helper Industries.

All in all, ConIX impressed me very favorably. I was unable to find any real bugs (and I carry a very powerful Murphy field around with me). The few items I found to disagree with me could legitimately be called personal pecadillos.

There are a couple of features I wish were there, of course; there always are. For instance, ARM can be patched to access a real-time clock if one is present in your system. I would like to see hooks for direct command-line access to the date and time as well.

Overall though, ConIX is a very fine piece of work that can significantly reduce the effort needed using a CP/M-based micro and also act as a vehicle for familiarization with more powerful operating systems.

By Daniel Lunsford

CSHARP Realtime Toolkit
CGRAPH

Hardware required: Any computer with a C compiler

Price: CSHARP—$600.00 (single user license), CGRAPH—$49.95

Available from: Systems Guild Inc., P.O. Box 1085, Kendall Square Station, Cambridge, Mass. 02142, (617)451-8479

Support: Full technical support for CSharp

Programming for industrial and scientific process control applications often begins with a state diagram, a graphical representation of a state system. Since the computer typically has to monitor numerous sensors and control various pieces of equipment in response to discrete and asynchronous inputs, a state system description is usually the best way to visualize the process.

Once a state diagram has been prepared, however, the task remains to convert it into working code by using an inherently unstructured maze of *if. . . then*, *while . . . do* and *goto* statements. Such code is tedious to write and subject to subtle errors that manifest themselves only after the software has been commissioned.

Ideally, a set of software tools should be available to allow the programmer to directly enter a description of the state system itself and let the computer simulate its operation in response to defined inputs. It would be even better if these same tools could handle the lower-level details of process control programming as well, such as developing code for the real-time scheduling of prioritized procedures, defining and monitoring hardware and software-initiated events, and defining hardware interrupts in a consistent, processor-independent manner.

Continuing with the wish list, why not ask that these software tools generate efficient C source code and that the source code for the tools themselves be available for custom modifications and extensions? Finally, let's request a complete real-time data processing utility for graphics display on a CRT or plotter.

Welcome—with a few minor limitations—to Systems Guild's CSHARP Realtime Toolkit.

To fully appreciate what CSHARP has to offer, you need to be proficient in C programming and also understand the architecture of your particular computer system. This software package is definitely not for the novice—it is a set of production tools for the professional working with real-time computer applications.

The Toolkit is divided into five distinct tools: CISR, CEVENT, CSCHED, CSTATE, and CGRAPH. Each can be used independently of the others and are extremely useful as such. However, the true power of CSHARP becomes most apparent when the tools are combined to create a complex set of state systems. In combination, they can be considered as a hierarchy:

CGRAPH -> CSTATE -> CSCHED ->
CEVENT -> CISR

CGRAPH works very well as a stand-alone software tool—so well that Systems Guild offers a version of it as a separate and inexpensive package. The discussion of CGRAPH at the end of this review shows in detail what you can expect from CSHARP. Here is a brief look at the other four tools.

CISR—interrupt servicing. CISR

consists of only two functions, *newisr()* and *oldisr()*, that are used to dynamically declare C procedures as interrupt handlers. This is an elegant concept that extends C's famed software portability to the hardware level. Interrupts are notoriously dependent upon the specifics of the machine and operating system. CISR hides these dependencies beneath a consistent C source level interface.

CEVENT—symbolic event handling. Moving up the hierarchy brings us to CEVENT. A state system will change states only in response to significant events. These can be hardware interrupts, inputs from sensors monitored by software polling loops, scheduled events activated by a real-time clock, or programmed software events. CEVENT offers seven functions to define and monitor these events in real time.

CSCHED—prioritized real-time scheduler. One of the hallmarks of process control programming is that events sometimes occur faster than the computer can process them. One solution to this problem is to enter the procedures that might generate events into a prioritized queue and respond to them in order of entry and priority. Once started, a procedure will execute through to completion unless interrupted by a higher priority procedure.

CSCHED consists of an executive designed to manage such a queue. It also allows you to specify the time interval at which a procedure is to be run and how many times it is to be repeated.

CSTATE—multiple state system control. CSTATE can be used to define and execute any number of state systems, limited only by the computer's available memory and processor speed. Execution of the simulated systems can be in response to events processed by CEVENT, execution of procedures under CSCHED, interrupts captured by CISR, or any other user-defined event. These systems are executed in parallel, prioritized according to the order in which they were defined.

The value of the CSHARP Realtime Toolkit is immeasurably enhanced by the availability of its source code. If you have a C compiler or cross compiler and an assembler for your target machine, you can run all of CSHARP's tools on it. If the tools are not to your liking, you are free to enhance and modify them as you wish.

Speaking of enhancements and modifications, the actual source code for CSHARP is surprisingly simple. Rather than equate size with monetary value, however, you should consider that simple code is often elegant and fast running. CSHARP is not an exception.

CSHARP relies heavily on the use of pointers to functions. This extremely powerful but often overlooked facility of C (even K&R's "The C Programming Language" discusses it almost in passing) allows CSHARP functions to pass user-defined procedures back and forth as easily as if they were data. If your programming expertise has been developed under FORTRAN or BASIC, study of CSHARP's source code should be rewardingly educational.

CSHARP is currently available for the 8088/86 and PDP/LSI-11 processor families, with Systems Guild working on support for the 6809/68000 family. If your target machine is not among these, conversion of the assembly language support routines for CISR should not be an insurmountable task—the PDP/LSI-11 source code covers two pages and the 8088/86, six pages. Free technical support, including assistance in porting to new machines, is available from Systems Guild for 90 days after date of delivery.

The C compilers used by Systems Guild were Computer Innovations' C86 and Lattice C (both large and small memory models) under PC/MS-DOS and Whitesmiths' C under RT-11. The IBM PC version of CSHARP comes with sample interrupt handlers and assembly language drivers for the IBM time-of-day clock (to drive CSCHED's executive) and both the IBM and Hercules color graphics boards.

The manual for CSHARP is well-written and accurate, if somewhat terse. Some circle and arc generation functions in the source code for CGRAPH were not documented. Systems Guild's development of the various tools is ongoing, and what you receive is their latest work, including undocumented (in the manual) features. Work in progress at Systems Guild (which will likely be finished by the time you read this review) includes a UNIX-compatible I/O library that will be compatible with both real-time data acquisition and the CGRAPH module and drivers for popular data acquisition boards such as the Data Translation DT2801 and Metrabyte Dash8.

One shortcoming of the CSCHED tool is that it does not permit events to be queued for processing, only procedures that might generate events. While in many cases this is not significant, it sometimes is essential to have the ability to queue the events themselves. Systems Guild reports that it is currently working toward this goal.

It would have been nice if Systems Guild could have provided a more sophisticated means of converting a state diagram into a state system. At present, CSHARP provides the data structures for the states, execution scheduler queue, events and interrupts, and executives to run them—you must provide the rest. It is hoped that future versions will include a graphically oriented state diagram editor for automatic generation of state systems. Various directed graph algorithms that can validate state systems, eliminate nondeterminism (possible transitions to two different states for the same event) and minimize their number of states. These could be potentially useful additions to CSHARP.

Is the CSHARP Realtime Toolkit worth the initial source license cost of $600? If you are involved in real-time work and do your programming in C, the answer is most likely yes. CSHARP can save you many programming manhours—how far does $600 go when you consider your time as a professional?

As a final comment on the CSHARP package, state systems are not limited to describing process control tasks. They also appear in compiler design and lexical analysis for natural language processing. Their use in other areas of problem solving appears to be underutilized. Software tools such as CSHARP may help us expand our horizons to include them in our repertoire of valued programming techniques.

CGRAPH. Supplied with the CSHARP package is the graphics module CGRAPH. At first glance, CGRAPH appears to be yet another collection of those simple graphic functions published in home computer magazines. In common with its public domain brethern, it offers such mundane commands as *dotat* (display a dot), *lineto* (draw a line), and *drawchar* (display a character). CGRAPH has no arc, circle, or ellipse generation routines, no area fills or crosshatch pattern functions. Nothing is exciting here, it would seem.

First glances deceive. On closer inspection, there turns out to be much more than this. CGRAPH is device independent, supports multiple simultaneous windows on each physical display unit, and if your OS allows multitasking through interruptible and reentrant tasks, CGRAPH supports it. Written in C and supplied in source code form, CGRAPH provides the programmer with a flexible and powerful set of graphic programming tools. Costing only $49.95, it is a software package that should be looked at carefully by anyone who knows C and is interested in computer graphics.

Graphic plotting commands. CGRAPH provides only seven graphic plotting functions, really a bare minimum for graphic programming. However, it is a fairly simple task (although not trivial) to extend these functions into a full set of graphic commands, including functions for drawing arcs, circles, ellipses, area fills and patterns.

Virtual display initialization. As a basic minimum, CGRAPH requires only that the graphic display device be capable of plotting a point at a specified position on its display surface. The user must write a C-callable function (using either C or assembly language) that provides the necessary interface to the hardware device. The function *setpoint()* is then used to declare a pointer to this interface (referred to as a *handler* in the CGRAPH documentation). Whenever a CGRAPH function needs to plot a point, it simply passes the coordinates of the point to the user-defined handler.

If the hardware device has built-in firmware to plot a line given only its endpoints or a character at the current pen position given only the character, then the functions *setline()* and *setchar()* can be used respectively to point to user-defined handlers for these features. If they are not available, CGRAPH does the necessary calculations in software and plots the lines and characters point by point.

Notice that you are not limited to one handler per program. *Setpoint()*, *setline()* and *setchar()* only point to the handlers. It is therefore a simple matter to call these functions with new handler names as arguments while the program is running to dynamically redefine the attributes of the virtual display device.

The functions *setchrsize()*, *setgrain()* and *setorient()* are more or less self-explanatory. Unfortunately, CGRAPH does not support the drawing of characters at any orientation, but only at right angles to the x and y axis. On the plus side, however, the internal character generator of CGRAPH maintains a shape table in memory. Since the software is provided as source code, it is an easy task to redefine the characters in the table to suit your needs. With a bit of source code juggling, you can even maintain multiple character sets in your application programs.

Virtual display definition. *Setview()*, *setwind()*, *clip()*, *remap()* and *setsta()* form the virtual display definition commands. It is possible to have more than one virtual display device active at one time. You can send the output of one virtual display to one or more physical devices, and you can display the output of several virtual displays on one physical device. You can also dynamically redefine the virtual display attributes during program operation by calls with new arguments to the virtual device definition and initialization function.

All of this magic is performed by storing the attributes of each virtual display device in a memory-resident data structure called a frame. Calls to the virtual display initialization functions are used to modify this data structure, and the function *setsta()* can be used to set up a new frame and return a pointer to the previous one. By doing so, you in effect define multiple virtual display devices.

CGRAPH can also be used with interrupts, with the interrupt service routine storing the current frame and activating another or creating a new one. Upon completion of the interrupt task, the old frame can be restored.

C compiler requirements. With all of the programming power that the preceding implies, you might think that CGRAPH is a very complex and lengthy piece of software. Surprisingly, it is anything but complex. Even with the *#include* and header files added on, CGRAPH.C takes only 12 pages to print out at 60 lines per page.

CGRAPH is written in Manx Software's Aztec CII for CP/M-80 and Computer Innovations' C86 for MS-DOS. These implementations of C closely follow the standard UNIX version 7 C so that CGRAPH is transportable to virtually any operating system/hardware combination that supports a C compiler. A few machine dependencies in the handler functions are used to output data directly to the hardware, but these generally have their equivalents in C compilers written for other operating systems.

Systems Guild is currently supplying CGRAPH in two formats: 8-in. SSSD disks for CP/M-80 and 5¼-in. disks for MS-DOS. (A version written for the BD Software C compiler is also available.) However, inquiries are invited for other formats and operating systems, and even in the worst case, you can always buy the CP/M or MS-DOS version, print out the source code, and reenter it by hand into your system. Twelve pages of C code is not a horrendous amount of data entry for the benefits returned.

Physical device interface requirements. One minor problem with CGRAPH is that you have to write your own physical device interface modules (handlers). To do this you have to know the details of your graphic display device's command set and how to address them through either assembly language or C code. Fortunately, Systems Guild has been thoughtful enough to provide example handlers (written in C) for the Epson

Realia COBOL

What to do while your COBOL programs compile and execute:

1. Wait.
2. Wait some more.
3. Stop waiting. Call Realia.

Patience isn't always a virtue.
Realia COBOL is **fast:**

Compilation Speed (minutes:seconds)

Lines in Program	Realia COBOL	mbp COBOL	Level II COBOL	R-M COBOL	Microsoft COBOL
1,000	:51	8:33	3:42	5:05	5:11
5,000	3:30	48:07	16:58	*	45:26

*Could not successfully compile the program.

Execution Time Ratio
(Gibson Mix; calculated S-Profile)

Realia COBOL	mbp COBOL	Level II COBOL	R-M COBOL	Microsoft COBOL
1.0	3.6	14.7	21.6	22.3

Sieve of Eratosthenes
0.818 seconds per iteration

All benchmark tests were performed on an IBM PC XT with 192KB of memory. IBM PC XT is a registered trademark of International Business Machines Corporation; mbp COBOL, of mbp Software and System Technology; Level II COBOL, of Micro Focus; R-M COBOL, of Ryan-McFarland; and Microsoft COBOL, of Microsoft.

Realia COBOL is written in COBOL. We offer you the tools we use ourselves:

- Our FOLLOW-THE-SOURCE™ interactive symbolic debugger. Works with normal native code.
- A speedy full-screen editor that handles very large files.
- Mainframe IBM VS COBOL compatibility.
- Interfaces to Assembler and C.
- No royalty or run-time fee.
- No limit on program size, up to available memory.
- In our new release, no need to insert the product diskette when you're using a hard disk.

Realia COBOL costs $995. Qualified companies can try it for free. Call us. And ask about our other products, Spacemaker™ and Termulator™.

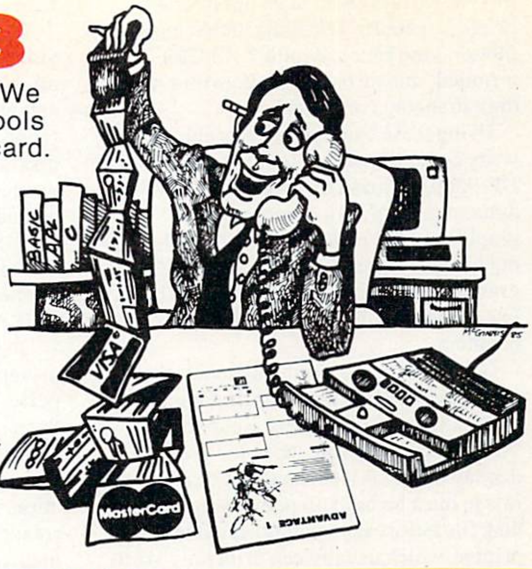
What are you waiting for?

REALIA inc.
10 South Riverside Plaza
Chicago, Illinois 60606
(312) 346-0642
TELEX: 332979 (REALIA INC)

CIRCLE 76 ON READER SERVICE CARD

ADVANTAGE #8

At Programmer's Connection we appreciate your business. We try to make it convenient for you to buy the programming tools you need. Some vendors penalize you for paying by credit card. At Programmer's Connection there is no extra charge on any prepaid order, and that includes paying by credit card. Furthermore, your account is not charged until your order is shipped. At Programmer's Connection there is no extra charge for convenience.



Discover the advantages of buying from Programmer's Connection:

1. We offer the latest version of a product.
2. Most popular products are in stock ready to be shipped.
3. Receive same manufacturer's support as if buying direct.
4. Experienced professional programmers are on staff.
5. Choose from a large selection of the best software products available.
6. Knowledgeable and courteous sales staff.
7. Significant discounts off of retail prices.
8. No extra charge on prepaid orders, including major credit cards.
9. Reasonable charges for shipping and handling.
10. Toll free services from Canada and the U.S.

NEW LISTINGS:

	List	Ours
Multi-Halo Graphics by Media Cybernetics	250	**199**
PANEL *Screen Designer ver. 6.0* by Roundhill	295	**234**
PANEL *6.0 updates*	50	**50**
Pasm86 Macro Assembler by Phoenix	295	**259**
Pfinish Performance Analyzer by Phoenix	395	**359**
Pmaker *Program Development Mgr.* by Phoenix	195	**179**
RM/Fortran by Ryan-McFarland	595	**439**
Turbo Pascal 3.0 by Borland	70	**59**

C terp Complete C Interpreter

Full K&R C interpreter/semi-compiler which can access functions and externals compiled on various C compilers. Comes with a powerful, built-in screen editor and a source-level debugger.

List Price $300 Our Price **$269**

C LANGUAGE:

	List	Ours
Computer Innovations C-86 Compiler	395	**299**
DeSmet C Compiler with Debugger	159	**145**
Lattice C Compiler from Lattice	500	**339**
Lattice C Compiler from Lifeboat	500	**299**
Mark Williams C Compiler w/Source Debugger	495	**429**
Run/C Interpreter by Age of Reason	150	**129**
Safe C Standalone Interpreter by Catalytix	400	**400**
Wizard C Compiler by Wizard Systems	450	**399**
Xenix Development System by SCO	1350	**1099**

Microsoft C Compiler version 3.0

This entirely new version of Microsoft's C Compiler features extremely fast and compact code generation, small, medium and large memory models, Xenix compatibility, a linker and a librarian.

List Price $395 Our Price **$339**

OTHER LANGUAGES:

	List	Ours
8088 Assembler w/Z-80 Translator 2500 AD	100	**89**
APL★Plus/PC by STSC	595	**469**
BetterBASIC by Summit Software	200	**169**
Modula-2/86 by Logitech	495	**439**
Professional BASIC by Morgan Computing	95	**89**

Microsoft Macro-Assembler version 3.0

New version of MASM-86 includes a symbolic debugger, object linker, librarian, cross reference utility, a MAKE facility and PC/AT support.

List Price $150 Our Price **$119**

C UTILITIES:

	List	Ours
C Power Paks from Software Horizons	Call	**Call**
C-Sprite *Symbolic Debugger for Lattice C*	175	**159**
c-tree by FairCom	395	**359**
C Utility Library by Essential Software	149	**119**
dBC *dBase/C Interface* by Lattice	250	**219**
ESP for C by Bellesoft	349	**279**
GraphiC by Scientific Endeavors	250	**209**
Greenleaf C Functions Library	185	**139**
Greenleaf Comm Library	185	**139**
Safe C Dynamic Profiler by Catalytix	150	**150**
Safe C Runtime Analyzer by Catalytix	400	**400**
Windows For C by Creative Solutions	195	**139**

Pre-C by Phoenix Software

A complete lint-like utility that helps detect logic errors by searching for inconsistencies in functions and data types across multiple files. A powerful addition to anyone's C toolbox.

List Price $395 Our Price **$339**

OTHER PRODUCTS:

	List	Ours
APL2C by Decision Images *Interfaces APL to C*	150	**139**
Blaise Tools for Pascal	Call	**Call**
Btrieve by SoftCraft	250	**199**
Codesmith-86 Debugger by Visual Age	145	**129**
Epsilon *Text Editor* by Lugaru	195	**179**
FORTRAN Libraries by Alpha Comp. Serv.	Call	**Call**
FORTRAN Scientific Subroutine Library	175	**159**
Periscope *Debugger* by Data Base Decisions	295	**269**
Pfix-86 Plus by Phoenix	395	**299**
Plink-86 *Overlay Linker* by Phoenix	395	**299**
Pmate *Macro Text Editor* by Phoenix	225	**159**
Polytron Products *We Carry a Full Line*	Call	**Call**
Profiler by DWB Associates	125	**89**
Screen Sculptor by Software Bottling	125	**109**
XTC *Text Editor* by Wendin	99	**89**
Xtrieve by SoftCraft	195	**169**

Prices are subject to change without notice.
Account is charged when order is shipped.

1-800-336-1166 In Canada: **1-800-225-1166**

Call for our Catalog

Programmer's Connection
136 Sunnyside Street
Hartville, Ohio 44632
(216) 877-3781 (In Ohio)

"Programmers Serving Programmers"

CIRCLE 23 ON READER SERVICE CARD

MX-series and Okidata 92 dot matrix printers, generic Tektronix and Selenar plotters, the Heath/Zenith Z-19 CRT terminal, and an oscilloscope with x-y digital/analog converters.

Using these examples, I was able to write custom handlers for an Anadex DP-9500 dot matrix printer and Microdynamics Corp. Microsprite S-100 graphics board in less than an hour. I do not consider myself to be an expert C programmer, so even the novice should have few difficulties in writing custom handlers.

Graphics programming demands that you have some means of storing images on disk files in a relatively compact form. This is especially true when the graphic display device is a dot-matrix printer; the image must be built up point by point in a disk file before being dumped to the printer, which usually can print only six to 10 horizontal rows of dots at a time. CGRAPH does provide a set of functions for creating and reading such files. However, no utilities are provided to edit these files or otherwise manipulate them.

The 40 pages of documentation provided are excellent. The program operation and functions are clearly described, and what is not covered in the documentation is described with equal clarity in the disk files. Even the source code is well-written and commented where necessary, an unusual occurrence for C program listings. Several examples included as disk files demonstrate the use of the various functions and the capabilities of CGRAPH.

On the negative side, I do have a few disappointments. Although CGRAPH has performed flawlessly for me, I wish that Systems Guild had included functions for some of the more complex graphic functions such as arc, circle and ellipse generation, area fills and crosshatching, image rotation, and so forth. The algorithms for them are well-known and not at all difficult to implement.

I would also like to have seen even a rudimentary editor for the graphics disk files, but for less than $50 that may be asking too much.

What you purchase with your money is a single user license. You can use and modify the source code as you see fit, but you can't resell or otherwise distribute the source or object code in any form without negotiating a secondary source license or object royalty license with Systems Guild. Systems Guild does not provide technical support for CGRAPH. However, given that the source code provided is so cleanly written and fully documented, this should not present any problem.

Minor gripes aside, I have this to say about CGRAPH: I like it very much. It is inexpensive for what it offers, the software is in C source code so that you can customize it to your heart's content and thoroughly understand how it works, and the functions can be compiled along with your own applications software to provide custom graphics programming packages. I recommend it.

By Ian Ashdown

Plink86

Hardware Requirements: One or more disk drives, 128K of memory, MS-DOS 2.0+ or CP/M 86
Price: $395
Available from: Phoenix Software, 1420 Providence Hwy., Suite 115, Norwood, Mass. 02062
Support: Updates ($35, with manual $50)

Plink86's advertised capabilities give a very favorable initial impression of the product. It can handle up to 4,095 overlays stacked up to 32 levels deep. Overlay reorganization can be accomplished without recompilation. Based on those claims alone, Plink86 outstrips all its competition.

The Plink86 package consists of a floppy, manual, and accompanying library manager PLIB. The floppy contains the linker itself (as .EXE or .CMD depending on the operating system), a checksum program (MS-DOS version only) to test the integrity of the linker, an overlay library file that contains run-time routines, and a number of test files. A few utilities are also included but are superfluous to the package itself.

The manual is the usual IBM three-ring affair, with no slipcover. It is divided into sections that include an explanation of a linker, Plink86's approach to the problem, installation and checkout, and a list of the commands in reference book format. A section of examples follows, with six appendices covering error messages, hints, and other points. The manual is not difficult to digest but is not quite up to the standards of larger corporations. It could certainly use a rewrite by a competent technical writer.

Plink86 is a two-pass linker. During the first pass, modules are read, and Plink86 determines which are to be loaded. The segment addresses are then assigned. During the second pass, the output file is created. While a two-pass system is slower than a single pass one, it does allow Plink86 to generate more efficient programs by having information about sizes and buffers predetermined.

It may be assumed that Plink86 will take considerably more time than one-pass linkers to perform its tasks. This is true, but the differences are not too important. Considering the power of Plink86 compared with MS-DOS LINK, the distinction is minor.

Plink86 can accept instructions either interactively or on one command line. The free-form approach of parsing is used, making the instructions independent of columns. Indentation can then be used to organize instructions. For large programs, it is usually ideal to create a file of the required statements with a word processor and then have Plink86 read that file. The instruction file is read with a "@" in front of the instruction file's name (for example, PLINK86 @INSTRUCT).

Plink86's commands cover almost every possibility of manipulation one could require from a linker. A simple file can be linked with the command *PLINK86 FI FILENAME*. The *FI* is an abbreviation of the *FILE* command, which gives the name(s) of the file(s) to be linked. Any Plink86 command can be abbreviated to a short form as long as it is unambiguous.

After processing, Plink86 issues a message that gives the executable files' names and the size memory required when run. Ouput files are named similarly to the input file, unless specified otherwise with an *OUTPUT* command. Thus:

```
PLINK86 OUTPUT TEST2.EXE
        FILE  TEST1.OBJ
```

will read the object file TEST1 and create the file TEST2.EXE. Libraries to be searched for routines are identified with the *LIBRARY* command.

Using these commands to link a file TEXT.OBJ with routines in TESTLIB.LIB, the instructions can be given as either:

```
PLINK86 OUT TEST.EXE FI TEST.OBJ
        LIB TESTLIB.LIB
```

or using the formatted approach:

```
PLINK86 @TEST
```

where the file TEST contains the following:

```
OUT TEST.EXE
FI TEST.OBJ
LIB TESTLIB.LIB.
```

The latter approach can be seen much more clearly, and the formatted instructions are readily understood. When command sequences approach the dozens, the formatted approach becomes almost mandatory for comprehension. Instructions can be appended after the formatted input approach in case an option is required that is not in the command file.

MS-DOS directory paths are supported by Plink86, and the path to be searched for files can be specified. An error message is generated if a file cannot be located.

The most important part of Plink86 remains the linking of modules in some order. Plink86 will follow a canonic program-data segment order, as given in the input commands, unless overridden by a specific *CLASS* command. (A class refers to the type of segment—data, program, etc.)

This situation can be illustrated by considering a simple program that contains no overlays. A single section is involved with all segments in their canonic order in the file. Plink86 will normally assign the segments in the same order that they appear in the file. Each segment will be allocated within memory in the order they are encountered, with program segments taking precedence over data segments.

A new section of a program can be constructed by the linker using the *SECTION* command. Following this instruction, any modules found by subsequent *FILE* or *LIBRARY/SEARCH* commands will be loaded into that section. The section can be named for use with the *MAP* command. Thus the series:

```
SECTION = SEC1
FILE TEST1, TEST2
SECTION = SEC2
FILE TEST3
```

will create two sections, SEC1 and SEC2. Segments from modules in the input files TEST1 and TEST2 will be assigned to section SEC1, and segments from modules in the file TEST3 will be assigned to section SEC2. A section command is thus required by all linking instructions, as the linker has to assign the modules to something. Plink86 will create a *SECTION* by itself at the start of the commands if the instruction is not given.

As mentioned earlier, the canonic ordering of the segments can be overridden with the *CLASS* command. This causes all segments in the class to be moved to the current section. A similar command is *MODULE*, which will assign all segments from the specified module to the current section. This is usually used to pull modules from a library. A final version is *GROUP*, which moves all segments within a group to the current section.

All these options may appear confusing at first glance, but essentially they are used to move varying sized blocks from one place to another in the linking process. They follow a hierarchy of *CLASS, MODULE,* and *GROUP,* with *CLASS* having the highest priority. These are then followed by the *FILE* and *LIBRARY* commands. (The instructions are only used when the ordering of the segments is to be different than encountered in the normal canonic system Plink86 uses.)

Overlaying sections is handled by another set of commands. An overlay area is a group of sections that will share the same address space. The overlay area is created with a *BEGINAREA* command and ended with an *ENDAREA* command. Sections that are defined (created) between these instructions then become overlays. The addresses of the overlays are all assigned by the linker.

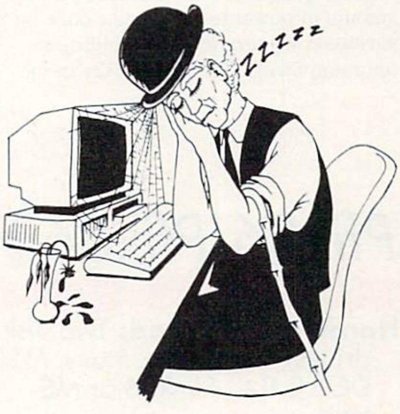
Several overlay areas can be created by using multiple *BEGINAREA* and *ENDAREA* commands. This is required with nesting applications. When nesting is used, Plink86 requires that the "parents" of the called overlay be in memory also. Otherwise a crash can occur.

Plink86 allows an additional feature that may be useful in some applications. Overlays (or groups of overlays) can be assigned to separate files. This assignment may be required if several floppies are involved. Then the overlay manager can prompt for the required disk with the necessary file(s) on it. Also, modularly programmed applications with different levels of support can be used as a marketing tool. (A graphics routine available as an optional extra can be sold as an overlay and called if available.)

Several considerations may be necessary, depending on the source code's origin, for Plink86 to use them properly. Any such considerations are discussed in detail in the manual.

For a programmer, one important aspect of the Plink86 package enhances it. The overlay loader supplied with Plink86 can be distributed with no license fee in any application that used Plink86. Plus the loader can be customized and distributed by the programmer at will. An appendix details the process of customization.

BORED?...
...waiting for C programs to compile and link?



Use **C-terp**
the **complete** C interpreter

This is the product you've been waiting (and waiting) for!

Increase your productivity and avoid agonizing waits. Get instant feedback of your C programs for debugging and rapid prototyping. Then use your compiler for what it does best...compiling efficient code ...slowly.

C-terp Features

- Full K&R C (no compromises)
- Complete built-in screen editor-- no half-way house, this editor has everything you need such as multi-files, inter-file move and copy, global searching, auto-indent, tab control, and much more.
- Fast--Linking and semi-compilation are breath-takingly fast. (From edit to run completion in a fraction of a second for small programs.)
- Convenient--Compiling and running are only a key-stroke or two away. Errors direct you back to the editor with the cursor set to the trouble spot.
- Object Module Support -- Access functions and externals in object modules produced by C86 or Lattice C or assembly language. Utilize your existing libraries unchanged!
- Complete Multiple Module Support-- Instant global searches, auto-compile everything that's changed, etc.
- Many more features including batch mode, 8087 support and symbolic debugging.
- Runs on IBM PC, DOS 2.x, 192K and up.
- **Price: $300.00 (Demo $45.00) MC, VISA**

Price of demo includes documentation and shipping within U.S. PA residents add 6% sales tax. Specify C86 or Lattice version.

GIMPEL SOFTWARE
3207 Hogarth Lane • Collegeville, PA 19426
(215) 584-4261

*Trademarks: C86 (Computer Innovations), Lattice (Lattice Inc.), IBM (IBM Corp.), C-terp (Gimpel Software)

The library manager supplied by Phoenix as part of the package makes standard libraries of modules easy to create and manipulate. A full set of commands for merging, creating, updating, adding, and deleting modules is available.

Overall, Plink86 is unique in that it performs a job that no other program does. It allows a programmer a seemingly endless amount of power to manipulate code into various configurations and determine optimum forms. The availability of the *MAP* function aids debugging enormously, especially on large programs where one can never be quite sure where a crash occurred.

Phoenix will support most compilers, including Microsoft C, Microsoft FORTRAN, Microsoft Pascal, Microsoft BASIC, Microsoft COBOL, Microsoft Assembler, Lattice C, Computer Innovations' C86, mbp COBOL, Mark Williams' C, RM FORTRAN, and IBM FORTRAN. The company has a policy of adapting Plink86 to each new compiler as it becomes available, and some compilers not on the "approved" list work with no problems at all. Phoenix's support of new versions will keep Plink86 current.

After using Plink86 a few times, it became very obvious that this was a product with no equal on the market. Plink86 should be a mandatory part of an application programmer's arsenal.

By Tim Parker

Pfix86/Pfix86 Plus

Hardware required: Two disk drives, 128K of memory, MS-DOS 2.0+, Plink86 or MS-Link
Price: $195 for Pfix86, $395 for Pfix86 Plus
Available from: Phoenix Software, 1420 Providence Hwy. Suite 115, Norwood, Mass. 02062
Support: Updates ($35, with manual $50)

Phoenix Software's entry in the competitive 8086 debugging software market is Pfix86 and its slightly enhanced Pfix86 Plus. Pfix86 is a multiple window debugger, allowing several breakpoint options, in-line assembly, data formatting, trap capabilities, and memory/register examination and modification. Pfix86 Plus combines with Phoenix's powerful Plink86 overlay linker to offer extra features for overlayed programs.

The Pfix86 package consists of a floppy containing the debugger, some sample files for demonstration purposes, and a standard IBM-sized manual laid out similarly to Plink86's manual. Unfortunately, it also had most of the same drawbacks as Plink86's manual. I found that I learned more about Pfix86's commands by reading through the demonstration than I did reading the manual.

Pfix86 uses a set of windows on the screen, with the window areas assigned to different functions: stack, CPU, file, breakpoints, data, and program. Only one of these areas is actually usable at one time, and they are selected by use of a function key or a command. The display is well thought-out, with each window outlined by a double line.

Once inside a window, the cursor location is shown by reverse video. The cursor can be moved with the IBM cursor keys. The Home, End, Pg Up and Pg Dn keys work in the windows (except for the CPU area).

Invoking commands in Pfix86 is easy. The structure of the program is set up so that simple mnemonics help do all the major commands. Most commands are accessible through a menu at the top of the screen. (Pfix86 reserves two lines at the top of the display for status messages, error messages, and the command menu.)

Menu commands are usually prefixed with the F1 function key. This produces a list of options at the top of the screen, in full word form. Typing the first letter of the required command invokes it. Alternatively, the desired command can be highlighted by moving the cursor to it (the active command appears in reverse video), and pressing return. Commands are not case dependent. After entering a command, that command's further submenu appears, and so on, until the entire command is completed with all required arguments. This approach makes it easy to use Pfix86, and the manual needs to be referred to only for more elaborate help.

Commands available cover the entire range of useful debugging features. Breakpoints can be set at any time in the code (except in ROM, of course), with up to 10 permanent and 10 temporary breakpoints active at one time. Breakpoints that are set (or disabled) are listed in the breakpoint window at the foot of the display screen and appear in the code in one of two ways.

Temporary breakpoints are easily enabled by pressing the F10 key when the cursor is on the line to be marked. An asterisk to the right of the instruction address in the program window indicates the breakpoint is set. If for some reason the breakpoint cannot be implemented, an error message appears.

Permanent breakpoints appear in the program window in half intensity. The manual claims they will be underlined, but that may be dependent on the monitor and configuration file. As the half intensity was easier to spot than an underlined statement, it was left as is.

Breakpoints can be cleared one at a time or all at once using an initialize command. They can be enabled or disabled singly or in a group.

In practice, the breakpoint facility was easy to use. The menu approach made it simple to flip the breakpoints on and off at will to test various code segments, and the easily set, temporary breakpoint facility was invaluable. The location of breakpoints was obvious in the program window by the half intensity, and the breakpoint window's list of all breakpoints helped enormously in keeping tabs on large programs.

Breakpoints are one of the most important features of any debugger, and Pfix86's breakpoint capabilities are superb. Most programmers will find themselves relying on the easily activated, temporary breakpoints quite a bit.

A number of disk functions are also available in Pfix86's command set. Naturally, programs can be loaded from disk, and data can be read and written to a file with a specified offset and file name. Absolute disk sectors can be read and written to memory and back to disk.

An evaluation feature allows an expression to be run through the arithmetic expression parser and the result displayed. The evaluation feature is useful for determining offsets and memory location in structured data files. Evaluation is also used for conditional breakpoints. The operators used in the evaluation function are the same as those used in the C language, with the same precedence rules. Using indirect addressing or specific memory addresses and the usual mnemonics for the 8086/8088 registers, expressions such as *[256] < (AX + BX − (5 / 9))* and *[256] = = GLOBALSYM* can be used as expressions in any location

Creators of COMPUTER LANGUAGE Sponsor the C Expert Forum

C Seminar/ Workshop

September 16-18
Cambridge, Mass

Never before have so many leaders in the C programming field gathered for one event. The C Seminar/Workshop will be an exciting forum on the latest technical innovations and C language developments. Best of all, you'll experience a practical, hands-on approach in small workshop sessions. All this in the beautiful autumn foliage of New England, just four blocks from Harvard Yard. The C Seminar/Workshop is brought to you by the publishers of COMPUTER LANGUAGE.

The cost for this comprehensive 2½ day event is only $695. Sign up by June 30th and receive a $100 early bird discount.

CURRICULUM

Speakers

Jim Brodie, *ANSI C committee chairman:* Overview of the ANSI Standardization Effort
P.J. Plauger, *author, ANSI C committee secretary:* Programming Style and C
Larry Rosler, *ANSI C language chairman:* Language Standardization Issues
Tom Plum, *author:* Efficiency of C Programs
Heinz Lycklama, */usr/group UNIX chairman:* UNIX Perspective on C
Leor Zolman, *compiler writer:* Porting C Programs between Operating Systems
Robert Ward, *C User's Group coordinator:* Structured Methods of Debugging C

Workshops

(Subject to change based on availability)

Seminar participants will be able to choose four from this list:

Debugging Techniques
Interpreters in a Development Environment
Programming for Portability
Efficient Code Generation
Cross Compilers
Network Data Base Theory and C
Object-File Formats for UNIX Systems
Philosophy and Methodology of Benchmarks
ANSI Standards: Questions & Answers
Code Readability and Organization
Asynchronous Communications
Writing Extensions to C
C / UNIX System Subroutine Interfaces
Porting C between CP/M, MS-DOS, and UNIX

C Seminar/Workshop Registration Form

Please enroll me in the C Seminar:

☐ Early Bird $595 (pay by 6/30/85)
☐ Single $695
☐ Multiple
(3 or more enrollments get $100 discount)
☐ I do not wish to enroll at this time but please send me more information.

Method of Payment:

☐ Check Enclosed
☐ Bill My Company

Make check payable to:
C.L. Publications Inc.

Name & title ____________________
Name & title ____________________
Name & title ____________________
Company ____________________
Address ____________________
City, State, Zip ____________________
Phone ____________________

DA85

COMPUTER LANGUAGE Seminar
131 Townsend St.
San Francisco, Calif. 94107
(415) 957-9353

CIRCLE 98 ON READER SERVICE CARD

where a data byte or word is normally used.

Memory manipulation can be accomplished with four commands. Memory can be cleared at a specified address for a similarly specified number of bytes. These are all set to zero. A set command allows a location or range of locations to be set to a list of bytes. The starting address, number of bytes to set, and the list to be placed there are specified in either hex or character string format. If the list to be set is shorter than the range specified, it is repeated. Assembly language mnemonic instructions can be entered, if enclosed in parentheses.

A sequence of instructions for actually running the program are necessary. An execute instruction with any command line arguments runs the program with a blanked screen, then after termination, prompts the user to hit any key to reenter the Pfix86 screen. This allows normal operation of DOS functions and programs from within Pfix86.

The analogous execution command *Go* allows the program to be run within Pfix86, starting at a specified address. A proceed command allows tracing and skipping of loops. Single instruction execution is followed, with some commands (such as *LOOP, JCXZ, CALL, INT* and some *REP*) skipped. This avoids iterative routines. The *Go* command can be activated either from a menu or a function key. A restart command starts the program execution at the current instruction pointer and can also be activated by a function key.

For debugging programs that access the screen, the Pfix86 package allows screens to be saved for later reference from the program. Then the Pfix86 window screen and the application program can be toggled between for comparison. Naturally, for graphic routines this is almost mandatory.

Neither versions of Pfix86 support the 8087 chip. This may be forthcoming from Phoenix, though, considering the relatively fast speed the company has shown with updates for its Plink86 linker.

At $195 for Pfix86 and $395 for Pfix86 Plus, they are among the higher priced debuggers (especially the Pfix86 Plus version). While symbols make debugging that much easier, unless a debugger was used on a regular schedule, consumers may balk at the enhanced version.

In summary, the Pfix86 package stacks up well against the other software debuggers on the market. The windows were pleasing to use and presented the best display of any debugger yet used by this reviewer. The need for multiple keystrokes to achieve a function (instead of one or two key sequences with most other debuggers) tended to be frustrating after a few hours of work, but the mnemonically organized structure means that control key sequences don't have to be memorized.

For the casual assembly language user, this product is probably ideal. For the regular debugger user, Pfix86 has to be considered among the best in its class. If Phoenix can follow with 8087 support and a rewritten manual, then Pfix86 may well garner the top spot.

By Tim Parker

A FULL C COMPILER FOR $49⁹⁵

The Ecosoft Eco-C88 compiler for the 8088 and MSDOS is going to set a new standard for price and performance. Consider the evidence:

Compiler	Eco-C88	Latice (1)	C86 (1)
Seive	13	11	13
Fib	44	58	46
Deref	13	13	--
Matrix	21	29	27
Price	$49.95	$500.00	$395.00

(1) **Computer Language,** Feb., 1985, pp.73-102. Reprinted by permission.

The Eco-C88 compiler is a full K&R C compiler that supports all data types and operators (except bit fields). Now look at the other features we offer:

★ 8087 co-processor support using a single library. If you install an 8087 later, the software will use it without having to recompile.
★ A robust standard library with over 150 functions, including transcendentals, color, and others.
★ OBJ output for linking with the MSDOS linker (LINK).
★ Error messages in English – no cryptic numbers to look up. A real plus especially if you're just getting started with C.
★ Easy-to-read and complete user's manual.
★ Works with all IBM and compatibles running MSDOS 2.0 (or later).
★ Plus many other features.

For $10.00 more, we will include the source code for the C library functions (excluding transcendentals). For an additional $15.00, we will include our ISAM file handler in OBJ format (as published in the **C Programmer's Library,** Que Publishing). The discount prices for the library source and ISAM only apply at the time the compiler is purchased. Please add $4.00 to cover postage and handling. To order, call or write:

Ecosoft Inc.
6413 N. College Avenue
Indianapolis, IN 46220
(317) 255-6476

ECOSOFT INC.

Eco-C (Ecosoft), MSDOS (Microsoft), UNIX (Bell Labs), CP/M (Digital Research), Z80 (Zilog), 8086, 8087, 8088 (Intel).

CIRCLE 17 ON READER SERVICE CARD

FAST SCREEN OUTPUT FOR TURBO PASCAL

FASTSCREEN ™
is a set of **inline** assembler and Pascal procedures for Turbo Pascal users.

- Display an entire screen or window almost instantly
- Process multi-field input screens that give your user full cursor control
- Sample program uses Conway's LIFE to illustrate use of procedures
- All source code included
- IBM PC XT AT and true compatibles

29⁹⁵ VA residents add 4% sales tax

TECHNISOFT
1710 Allied Street, Suite 37
Charlottesville, VA 22901
(804) 979-6464

Turbo Pascal is a registered trademark of Borland International

CIRCLE 31 ON READER SERVICE CARD

Thunder Software

- **The THUNDER C Compiler** - Operates under the APPLE Pascal 1.1 or 1.2 operating system or ProDOS. Create fast native 6502 programs to run as stand alone programs or as subroutines to Pascal programs. A major subset of the C defined by K & R. Includes a users guide, newsletters, Macro preprocessor, runs on APPLE][,][+, //e, //c. Source code for libraries is included. **Only $49.95**
- **ASSYST: The Assembler System** - A complete 6502 editor/ assembler and lister for APPLE DOS 3.3 or ProDOS. Menu driven, excellent error trapping, users guide, demo programs, source code for all programs! Great for beginners. **Only $29.95**
- **THUNDER XREF** - A cross reference utility for APPLE Pascal 1.1. XREF generates cross references for each procedure. **Only $19.95**
- **PDL** - Pascal Development Library. SCREENIO, DISKIO, CONVERSIONS, GRAPHICS, PLOT, MISC Units for Pascal programming. A must. **Only $29.95**
- **PROPLOT** - Easy and powerful graphing package for plotting data. **Only $24.95**
- **LINKIT** - Structured APPLESOFT processor. No more line numbers. **Only $39.95**

POB 31501 Houston Tx. 77231
713-728-5501
Visa, Mastercard, COD, checks accepted
Add $3.00 for P&H

CIRCLE 30 ON READER SERVICE CARD

STRUCTURE FOR BASIC

With PROFESSIONAL PROGRAMMING ENVIRONMENT

ELIMINATES LINE NUMBERS

ALLOWS MULTI-LINE CONDITIONALS

PCDOS/MSDOS

BENDORF ASSOCIATES
6006 S. MAIN
P.O. BOX 5910
ROSWELL, NM
88201
505 347-5701

VISA/MASTERCARD

WORKS WITH BASICA INTERPRETER COMPILER

FULL ERROR LOGGING

PROGRAM LISTER

LABELED PROCEDURES

MACROS

SUB-ROUTINES

LIBRARIES

$49.95

CIRCLE 6 ON READER SERVICE CARD

MINNESOTA

SNOBOL4

ONLY $44.95

This exceptional language will amaze you with its versatility. Compatible with main-frame SNOBOL4, use it at work and home. Perform those complicated programming jobs and prototype your exotic ideas quickly. This robust implementation supports large memory model, 8087(if float desired), 32 bit integers, 32K strings. Includes 60 page reference guide and sample programs including ELIZA. You can also get Griswold's et al definitive "green" book or even the source code. Needs IBM PC or 8088/86, ≥128K, IBM or MS DOS.

Guide + 5¼" SSDD diskette $44.95
Guide + diskette + "green" book........ $59.95
"Green" book only...................... $24.95
Source code and license $500.00
Prices postpaid. In NY state add sales tax.

☎ (914)271-5855

BERSTIS INTERNATIONAL
P.O. Box 441
MILLWOOD, NY 10546 USA

CIRCLE 45 ON READER SERVICE CARD

Hyperon Software

Specializing in innovative programming tools.

- Complete documentation and C-source provided (presently DOS only).
- Reasonable prices.
- High quality and good performance.

Products currently available:

C Preprocessor. Features include variables and expressions, loops, and full macros. Price — $39.95.

General purpose editor. Line oriented commands with a screen oriented submode. Command window. Price — $29.95

Order from:

HYPERON SOFTWARE
P.O. Box 3349
Costa Mesa, CA 92628

Enclose check or money order. California residents add 6%.
2532 Orange Ave., Costa Mesa, CA

CIRCLE 51 ON READER SERVICE CARD

NEW!

Advanced Trace86™

Symbolic Debugger & Assembler Combo

- Full-screen trace with single stepping; Even backstepping!
- Write & Edit COM & EXE programs
- Conditional breakpoints (programmable)
- Switch between trace and output screen; Or set up two monitors
- 8087, 80186, 80286, 80287 support
- Write labels & comments on code
- Polish hex/decimal calculator
- and more . . . Priced at $175.00

To order or request more information contact:

Morgan Computing Co., Inc.
800/527-0009
P.O. Box 112730, Dallas, TX 75011
(214) 245-4763

CIRCLE 22 ON READER SERVICE CARD

C Users' Group

Over 40 volumes of public domain software including:

- compilers
- editors
- text formatters
- communications packages
- many UNIX-like tools

Write or call for more details

The C Users' Group
415 E. Euclid • Box 97
McPherson, KS 67460
(316) 241-1065

CIRCLE 5 ON READER SERVICE CARD

OPT-TECH SORT™

SORT/MERGE program for IBM-PC, XT & AT

Now also sorts dBASE II files!

- Written in assembly language for **high performance** Example: 4,000 records of 128 bytes sorted to give key & pointer file in 30 seconds. **COMPARE!**
- Sort ascending or descending on up to nine fields
- Ten input files may be sorted or merged at one time
- Supports many file structures & data types
- Filesize limited only by your disk space
- Output file can be full records, keys or pointers
- Can be run from keyboard or as a batch command
- Can be called as a subroutine to many languages
- Easy to use — Fully documented
- **$99** — VISA, M/C, Check, Money Order, COD, or PO Quantity discounts and OEM licensing available

To order or to receive additional information write or call:

OPT-TECH DATA PROCESSING
P.O. Box 2167 Humble, Texas 77347
(713) 454-7428

Requires DOS, 64K and One Disk Drive

CIRCLE 62 ON READER SERVICE CARD

FoxBASE™

Interpreter/Compiler

- dBASE II® source compatible
- Runs 3-20 times faster than dBASE II
- 8087 coprocessor support
- 14 digit precision
- Up to 48 fields per record
- Full type-ahead capabilities
- Provides compact object code and program security
- Twice as many memory variables as dBASE II

FOX SOFTWARE INC.
13330 Bishop Road, P.O. Box 269
Bowling Green, OH 43402
419-354-3981

CIRCLE 29 ON READER SERVICE CARD

YOU CAN C

VIEW v3

The ultimate CP/M disk utility.

- **Edits** any disk sector
- **Displays** allocation bit map
- **Rebuilds** files automatically from selected blocks

If you ever had to recover a crashed disk, you need **VIEW**. Use **VIEW** to patch damaged sectors, un-erase files, examine other disk formats, more

COMPLETE C SOURCE CODE **$59**

BD Software's C Compiler

The most popular CP/M C Compiler comes with **symbolic debugger** and special Western Wares' **ISIS-II** function library **$140**

C-PACK

A disk full of useful CP/M utilities written in C, PEPE, DEL, BACKUP, DPATCH, ECHO, CHX, PAUSE, more Source incl **$19**

Other C Products

EZZAP, ICX, MEDIT, more
Send for a catalog today

CP/M®, Digital Research
ISIS-II ®, Intel Corp

Western Wares **303-327-4898**
Box C • Norwood, CO 81423

CIRCLE 41 ON READER SERVICE CARD

FORTRAN PROGRAMMERS

Discover why you should be using

F77L

the complete implementation of the ANSI FORTRAN 77 Standard for the IBM PC and compatibles.

If you are serious about your FORTRAN programming, you should be using F77L.

$477

Lahey Computer Systems, Inc.
31244 Palos Verdes Drive West, Suite 243
Rancho Palos Verdes, California 90274
(213) 541-1200

Serving the FORTRAN community since 1969

CIRCLE 15 ON READER SERVICE CARD

ADVERTISER INDEX

	PAGE NO.	CIRCLE NO.
AGS Computers Inc.	1	7
Amber Systems	8	2
Automata Design Assoc.	79	1
BD Software	76	4
Bendorf & Associates	95	6
Bertis International	95	45
Blaise Computing Inc.	73	8
Borland International	6,7	9
Brady Communications Co.	10	20
C Users Group	95	5
C Ware	14	11
CCA Uniworks Inc.	25	60
Chalcedony Software	69	43
Computer Helper Industries	64	10
Computer Language C Seminar	93	98
Creative Solutions	38	14
Ecosoft	94	17
Edward Ream	64	—
Entelekon	68	50
Essential Software Inc.	72	33
Fox Software	95	29
Gimpel Software	85	—
Gimpel Software	91	—
Greenleaf Software Inc.	84	44
HSC Inc.	56	12
Harvard Softworks	55	47
Human Computing Resources	32	93
Hyperon Software	95	51
Inmos Corp.	31	99
Introl Corp.	60	32
Lahey Computer Systems Inc.	95	15
Lattice Inc.	62	18
Lifeboat Associates	22	87
Lifeboat Associates	19	85
mbp Software and Systems Technology	57	53
Major Software	63	16
Manx Software Systems	cover III	69
Megamax Inc.	64	13
Micro-Software Developers	79	54
MicroTech Research Inc.	4	36
Micromotion	82	64
Microsoft Corp.	cover IV	—
Miller Microcomputer Service	29	37
Mindbank Inc.	75	63
Morgan Computing Co.	95	22
Mystic Canyon Software	74	57
Nantucket	2	59
Next Generation Systems	30	61
Op Tech Data Processing	95	62
Phoenix Computer Products Corp.	48,49	65
Plu Perfect Systems	20	35
Polytron Corp.	17	34
Poor Person Software	64	67
Programmer's Connection	89	23
Programmer's Shop (The)	12	40
Programmer's Shop (The)	58	94
RR Software Inc.	cover II	58
Rational Systems Inc.	24	72
Realia Inc.	88	76
SLR Systems	69	73
Soft Craft Inc.	78	38
Softaid Inc.	79	78
Software Horizons	63	25
Solution Systems	61	21
Solution Systems	61	95
Solution Systems	61	96
Solution Systems	61	97
Southwest Data Systems	80	46
Sparry Software Labs	80	66
Spruce Technology Corp.	77	26
SuperSoft	40	74
SuperSoft	82	77
SuperSoft	63	79
Systems Guild	20	27
Technisoft	94	31
Thunder Software	94	30
UniPress	39	81
Watcom Products Inc.	70	24
Western Ware	95	41
Whitesmiths Ltd.	50	39
Wizard Systems	69	86
Workman & Associates	80	68
XYLOGIC	23	28
Zeducomp	80	83

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

ADVERTISE

in the August issue of

COMPUTER LANGUAGE

Special
Exotic Language issue

Includes a Forth product wrap-up

Reservation deadline: June 5th

Contact:
Carl Landau
COMPUTER LANGUAGE
131 Townsend St.
San Francisco, Calif. 94107
(415) 957-9353

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

FREE INFORMATION

Name ______________________________

Company ______________________________

Address ______________________________

City, State, Zip ______________________________

Country ____________ Telephone number ______________________

June issue. Not good if mailed after October 31, 1985.

Circle numbers for which you desire information.

1	11	21	31	41	51	61	71	81	91	101	111	121	131	141
2	12	22	32	42	52	62	72	82	92	102	112	122	132	142
3	13	23	33	43	53	63	73	83	93	103	113	123	133	143
4	14	24	34	44	54	64	74	84	94	104	114	124	134	144
5	15	25	35	45	55	65	75	85	95	105	115	125	135	145
6	16	26	36	46	56	66	76	86	96	106	116	126	136	146
7	17	27	37	47	57	67	77	87	97	107	117	127	137	147
8	18	28	38	48	58	68	78	88	98	108	118	128	138	148
9	19	29	39	49	59	69	79	89	99	109	119	129	139	149
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150

I obtained this issue through:
☐ Subscription ☐ Passed on by associate
☐ Computer Store ☐ Other ____________
☐ Retail outlet

Comments ______________________________

____

____

____

____

Attn: Reader Service Dept.

2/6

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

FREE INFORMATION

Name ______________________________

Company ______________________________

Address ______________________________

City, State, Zip ______________________________

Country ____________ Telephone number ______________________

June issue. Not good if mailed after October 31, 1985.

Circle numbers for which you desire information.

1 11 21 31 41 51 61 71 81 91 101 111 121 131 141
2 12 22 32 42 52 62 72 82 92 102 112 122 132 142
3 13 23 33 43 53 63 73 83 93 103 113 123 133 143
4 14 24 34 44 54 64 74 84 94 104 114 124 134 144
5 15 25 35 45 55 65 75 85 95 105 115 125 135 145
6 16 26 36 46 56 66 76 86 96 106 116 126 136 146
7 17 27 37 47 57 67 77 87 97 107 117 127 137 147
8 18 28 38 48 58 68 78 88 98 108 118 128 138 148
9 19 29 39 49 59 69 79 89 99 109 119 129 139 149
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150

I obtained this issue through:
☐ Subscription ☐ Passed on by associate
☐ Computer Store ☐ Other ____________
☐ Retail outlet

Comments ______________________________

____

____

____

____

Attn: Reader Service Dept.

2/6

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 27346 PHILADELPHIA, PA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

P.O. BOX 11747
PHILADELPHIA, PA 19101

SUBSCRIBE

COMPUTER
LANGUAGE

Subscribe to COMPUTER LANGUAGE today for only $24.95—over 30% savings off the single copy price.

☐ Yes, start my Subscription to COMPUTER LANGUAGE today. The cost is only $24.95 for 1 year (12 issues).

☐ I want to increase my savings even more—send me 2 years (24 issues) of COMPUTER LANGUAGE for only $39.95.

☐ Payment enclosed ☐ Bill me

Name ______________________________

Company ______________________________

Address ______________________________

City, State, Zip ______________________________

Please allow 6–8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Canada orders $30.95 per year. Outside the U.S., $36.95/year for surface mail or $54.95/year for airmail.

BIZ5

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 27346 PHILADELPHIA, PA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

P.O. BOX 11747
PHILADELPHIA, PA 19101

SUBSCRIBE

COMPUTER
LANGUAGE

Subscribe to COMPUTER LANGUAGE today for only $24.95—over 30% savings off the single copy price.

☐ Yes, start my Subscription to COMPUTER LANGUAGE today. The cost is only $24.95 for 1 year (12 issues).

☐ I want to increase my savings even more—send me 2 years (24 issues) of COMPUTER LANGUAGE for only $39.95.

☐ Payment enclosed ☐ Bill me

Name ______________________________

Company ______________________________

Address ______________________________

City, State, Zip ______________________________

Please allow 6–8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Canada orders $30.95 per year. Outside the U.S., $36.95/year for surface mail or $54.95/year for airmail.

BIZ5

AZTEC The Most Powerful C

for the IBM AT • MACINTOSH • MS DOS • CP/M-80 • ROM APPLICATIONS
IBM PC/XT • APPLE // • CP/M-86 • TRSDOS • CROSS DEVELOPMENT

Why Professionals Choose Aztec C

AZTEC C compilers generate fast, compact code. AZTEC C is a sophisticated development system with assemblers, debuggers, linkers, editors, utilities and extensive run time libraries. AZTEC C is documented in detail. AZTEC C is the most accurate and portable implementation of C for microcomputers. AZTEC C supports specialized professional needs such as cross development and ROM code development. MANX provides qualified technical support.

AZTEC C86/PRO
— for the IBM AT and PC/XT

AZTEC C86/PRO provides the power, portability, and professional features you need to develop sophisticated software for PC DOS, MS DOS AND CP/M-86 based microsystems. The system also supports the generation of ROM based software for 8088/8086, 80186, and 80286 processors. Options exist to cross develop ROM code for 65xx, 8080, 8085, and Z80 processors. Cross development systems are also available that target most micro computers. Call for information on AZTEC C86/PRO support for XENIX and TOPVIEW.

POWERFUL — AZTEC C86/PRO 3.2 outperforms Lattice 2.1 on the DHRYSTONE benchmark 2 to 1 for speed (17.8 secs vs 37.1) while using 65% less memory (5.8k vs 14k). The AZTEC C86/PRO system also compiles in 10% to 60% less time and supports fast, high volume I/O.

PORTABLE — MANX Software Systems provides **real** portability with a family of compatible AZTEC C software development systems for PC DOS, MS DOS, CP/M-86, Macintosh, CP/M-80, APPLE //+, //e, and //c (NIBBLE - 4 apple rating), TRSDOS (80-MICRO - 5 star rating), and Commodore C64 (the C64 system is only available as a cross compiler - call for details). AZTEC C86/PRO is compatible with UNIX and XENIX.

PROFESSIONAL — For professional features AZTEC C86/PRO is unparalleled.

- Full C Compiler (8088/8086 - 80186 - 80286)
- Macro Assembler for 8088/8086/80186/80206
- Linkage Editor with ROM support and overlays
- Run Time Libraries - object libraries + source DOS 1.x; DOS 2.x; DOS 3.x; screen I/O; Graphics; UNIX I/O; STRING; simulated float; 8087 support; MATH; ROM; CP/M-86
- Selection of 8088/8086, 80186, or 80286 code generation to guarantee best choice for performance and compatibility
- Utility to convert AZTEC object code or libraries to Microsoft format. (Assembly + conversion takes less than half the time as Microsoft's MASM to produce MS object)
- Large memory models and sophisticated memory management
- Support products for graphics, DB, Screen, & ...
- ROMable code + ROM support + separate code and data + INTEL Hex Converter
- Symbolic Debugger & Other Utilities
- Full Screen Editor (like Vi)
- CROSS Compilers are available to APPLE //, Macintosh, CP/M-80, TRSDOS, COMMODORE C64, and ROM based 65xx, and 8080/8085/Z80
- Detailed Documentation

AZTEC C86/PRO-AT $500
(configured for IBM AT - options for 8088/8086)

AZTEC C86/PRO-PC/XT $500
(configured for IBM PC/XT - options for 80186/80286)

AZTEC C86/BAS includes C compiler (small model only), 8086 MACRO assembler, overlay linker, UNIX, MATH, SCREEN, and GRAPHICS libraries, debugger, and editor.

AZTEC C86/BAS	$199
AZTEC C86/BAS (CP/M-86)	$199
AZTEC C86/BAS (DOS + CP/M-86)	$299
UPGRADE to AZTEC C86/PRO	$310
C-TREE Database with source	$399
C-TREE Database (object)	$149

CROSS COMPILERS

Cross Compilers for ROM, MS DOS, PC DOS, or CP/M-86 applications.

VAX - > 8086/80xxx cross	$5000
PDP-11 - > 8086/80xxx cross	$2000

Cross Compilers with PC DOS or CP/M-86 hosts are $750 for the first target and $500 for each additional target. Targets: 65xx; CP/M-80; C64; 8080/8085/Z80; Macintosh; TRSDOS; 8086/8088/80186/80286; APPLE //.

AZTEC C68K
— for the Macintosh

For power, portability, and professional features AZTEC C68K-c is the finest C software development system available for the Macintosh.

The AZTEC C68K-c system includes a 68000 macro assembler, a linkage editor, a source editor, a mouse based editor, a SHELL development environment, a library of UNIX I/O and utility routines, full access and support of the Macintosh TOOLBOX routines, debugging aides, utilities, make, diff, grep, TTY simulator with upload & download (source supplied), a RAM disk (for 512K Mac), a resource maker, and a **no royalty** license agreement. Programming examples ar included. (Over 600 pages of documentation).

AZTEC C68K-c requires a 128K Macintosh, and two disk drives (frugal developers can make do with one drive). AZTEC C68K supports the 512K Macintosh and hard disks.

AZTEC C68K-c (commercial system)	$500
AZTEC C68K-p (personal system)	$199
AZTEC C68K-p to AZTEC C68K-c upgrade	$310
Mac C-tree database	$149
Mac C-tree database with source	$399
Lisa Kit (Pascal to AZTEC C68k object converter)	$ 99

AZTEC C65
— for the APPLE //

"...The AZTEC C-system is one of the finest software packages I have seen..." NIBBLE review, July 1984.

The only commercial C development system available that runs native on the APPLE II+, IIc, and IIe, the AZTEC C65 development system includes a full floating point C compiler compatible with UNIX C and other MANX AZTEC C compilers, a 6502 relocating assembler, a linkage editor, a library utility, a SHELL development environment, a full screen editor, UNIX I/O and utility subroutines, simple graphics, and screen functions.

AZTEC C65 (Apple DOS 3.3)	$199
AZTEC C65/PRO (Apple DOS + ProDos)	$350

(call for availability)

AZTEC C II/PRO
— for CP/M-80

The first member of the AZTEC C family was the CP/M-80 AZTEC C compiler. It is "the standard" compiler for development on CP/M-80. The system includes the AZTEC C II C compiler, an 8080 assembler, a linkage editor, an object librarian, a full library of UNIX I/O and utility routines, CP/M-80 run time routines, the SMALL library (creates modules less than 3K in size), the fast linker for reduced development times, the ROM library, RMAC and M80 support, library source, support for DRI's SID/ZSID symbolic debugger, and more.

AZTEC C II/PRO	$349
AZTEC CII/BAS	$199
C-TREE Database with source	$399
C-TREE Database in AZTEC object form	$149

AZTEC C80
— for TRSDOS (Radio Shack Model III & 4)

"I've had a lot of experience with different C compilers, but the Aztec C80 Compiler and Professional Development System is the best I've seen." 80-Micro, December, 1984, John B. Harrell III

This sytem has most of the features of AZTEC C II for CP/M. It is perhaps the best software development system for the Radio Shack Model III and IV.

AZTEC C80 model 3 (no floating point)	$149
AZTEC C80 model 4 (full)	$199
AZTEC C80/PRO (full for model 3 and 4)	$299

To order or for information call:

800-221-0440

(201) 530-7997 (NJ and outside U.S.A.). Or write: MANX SOFTWARE SYSTEMS, P.O. Box 55, Shrewsbury, N.J. 07701.

MANX

SHIPPING INFORMATION - Standard U.S. shipment is UPS ground (no fee). In the U.S. one day shipment is $20, two days is $10. Canadian shipment is $10. Two days shipment outside the U.S. is by courier and is freight collect.

TRS 80 RADIO SHACK TRS DOS is a trademark of TANDY.
APPLE DOS MACINTOSH is a trademark of APPLE.

For Technical Support
(Bug Busters) call: 201-530-6557

CIRCLE 69 ON READER SERVICE CARD

New Version

Sizzling C.

The fastest C. The C that Microsoft developed to write its own software programs. Hot.

So hot that we can make this claim: Virtually every program runs faster with Microsoft® C Compiler than with any other MS-DOS® C compiler.

Efficient C.

We give you everything you need to write code so tight your computer will scream.

"Preliminary testing on the Microsoft C Compiler produced code that was significantly smaller than that produced by other C compilers."
Paul Springer, Ashton-Tate.

"We found the FAR pointer very helpful for situations where a mix of memory models offers the greatest efficiency."
Robert Frankston, Software Arts.

"The portability of the code between MS-DOS and XENIX® is great."
Jim Bean, Peachtree Software.

But it's really no surprise that our C stretches your micro to its limits. We wrote both the MS-DOS and the XENIX operating systems.

For the name of your nearest Microsoft dealer, or to upgrade from Microsoft C Compiler or Lattice C, call (800) 426-9400. In Washington State, Alaska, Hawaii and Canada, call (206) 828-8088.

MICROSOFT.
The High Performance Software™

And make your programs really cook.

Microsoft C Compiler Version 3.0

Microsoft C Compiler

- Produces compact code and fast executables.
- Implements register variables.
- Small, Medium and Large Memory model Libraries-Mix models with NEAR and FAR pointers.
- Transport source and object code between MS-DOS & XENIX operating systems.
- Library routines implement most of UNIX System V C library.
- Choose from three Math libraries and generate in-line 8087/287 instructions or floating point calls.
  - —Floating point emulator (utilizes the 8087/287 if installed).
  - —8087/287 coprocessor support.
  - —Alternate math package—extra speed on systems without an 8087/287.
- Link routines written in Microsoft FORTRAN (V 3.3 or higher), Microsoft Pascal (V 3.3 or higher) or Microsoft Macro Assembler.
- Supports MS-DOS pathnames and Input/Output redirection.
- File sharing and record and file locking is supported.
- Do source level debugging, with the Symbolic Debug Utility, available separately with the Microsoft Macro Assembler Package.*

Library Manager

Create, organize and maintain your object module libraries created with Microsoft languages.

Object Code Linker

- Simple overlay linker combines relocatable object modules created using Microsoft Languages into a single program.
- Link very large programs (over 1MB, using overlays).

EXEPACK Utility

A new utility to compress sequences of identical characters from an executable file and to optimize the relocation table.

EXEMOD Utility

A new utility used to modify the fields in the header according to the instructions given by the user in the command line.

'C' Benchmarks—done on a Compaq Plus with 512k memory with no 8087. Program "SIEVE," with register variables,

	Exec Time	Code Size	EXE Size
Microsoft C	:9.39	141	5,914
Lattice C	:12.24	164	20,072

*Purchase both Microsoft C Compiler and Microsoft Macro Assembler and get a $25 rebate direct from Microsoft. See package for details.

Microsoft, MS-DOS and XENIX are registered trademarks and The High Performance Software is a trademark of Microsoft Corporation.